

1 Chapter 2  
2 **Overview**

3 Most ultra-dependable real-time computing architectures developed  
4 in the past have been specialised to meet the particular requirements of the  
5 application domain for which they were targeted. This specialisation has led  
6 to very costly, inflexible, and often hardware-intensive solutions that, by  
7 the time they are developed, validated and certified for use in the field, can  
8 already be out-of-date in terms of their underlying hardware and software  
9 technology. This problem is exacerbated in some application domains since  
10 the systems in which the real-time architecture is embedded may be  
11 deployed for several decades, i.e., almost an order of magnitude longer  
12 than the typical lifetime of a generation of computing technology.

13 The end-user companies in the GUARDS consortium all currently  
14 deploy ultra-dependable real-time embedded computers in their systems,  
15 but with very different requirements and constraints resulting from the  
16 diversity of their application domains: nuclear submarine, railway and  
17 space systems. The architecture developed by the project aims to  
18 significantly decrease the lifecycle costs of such embedded systems. The  
19 intent is to be able to configure *instances* of a generic architecture that can  
20 be shown to meet the very diverse requirements of these (and other) critical  
21 real-time application domains. A three-pronged approach is being followed  
22 to reduce the cost of validation and certification of instances of the  
23 architecture: a) design for validation, so as to focus validation obligations

1 on a minimum set of critical components; b) re-use of already-validated  
2 components in different instances; and c) the support of software  
3 components of different criticalities.

4 This overview chapter is structured as follows. Section 2.1 sketches  
5 the rationale for the design of the generic architecture, which is then  
6 summarized in Section 2.2. Central to the architecture is an inter-channel  
7 communication network, which is described in Section 2.3. Section 2.4  
8 details the inter-channel fault-tolerance mechanisms while Section 2.5  
9 discusses the scheduling issues raised by active replication of real-time  
10 tasks. Sections 2.7 and 2.8 discuss respectively the development and  
11 validation environments that accompany the architecture. Finally, Section  
12 2.9 describes the current prototypes.

## 13 **2.1 Design Rationale**

14 To merit the epithet “generic”, the architecture must be able to meet the  
15 widest possible spectrum of dependability and real-time requirements. To  
16 this end, we first consider some key non-functional requirements of typical  
17 applications in each of the three end-user domains. We then discuss the  
18 issues of fault classes and real-time scheduling.

### 19 **2.1.1 Key Non-Functional Requirements**

20 A typical instance of the architecture in the railway domain would be a fail-  
21 safe control system. Standards in this domain dictate extremely low  
22 catastrophic failure rates for individual subsystems (e.g., less than  
23  $10^{-11}$ /hour with respect to physical faults). In railway applications, it is  
24 common to physically segregate subsystems responsible for vital (safety-  
25 critical) functions from non-vital functions. We decided to investigate the  
26 possibility of a single instance supporting both high-integrity vital functions  
27 and low-integrity non-vital functions.

28 In the nuclear submarine domain, an instance of the architecture  
29 would typically be used to support secondary protection functions, which

1 are required to be ready to react in case of (rare) incidents. Two  
2 requirements from this application domain impose quite severe restrictions  
3 on the design space. First, it must be possible to separate redundant  
4 elements of the architecture by several meters so as to tolerate physical  
5 damage. Second, to avoid obsolescence during the submarine's lifetime,  
6 the use of unmodified commercial off-the-shelf operating system(s) is  
7 mandatory.

8 A particularly challenging application in the space domain is that of  
9 an autonomous spacecraft carrying out missions containing phases that are  
10 so critical that tolerance of several faults may be required (e.g., target fly-by  
11 or docking). During non-critical phases, the redundant elements may be  
12 powered down to save energy. Moreover, it is necessary for an instance to  
13 be able to support software of different integrity levels: high-integrity  
14 critical software that is essential for long-term mission reliability and  
15 potentially unreliable payload software.

### 16 **2.1.2 Fault Classes**

17 The architecture aims to tolerate permanent and temporary physical faults  
18 (of both internal and external origins) and should provide tolerance or  
19 confinement of software design faults. This wide spectrum of fault classes  
20 [Laprie, 1995 #2140] has several consequences beyond the basic physical  
21 redundancy necessary to tolerate permanent internal physical faults.  
22 Tolerance of permanent external physical faults (e.g., physical damage)  
23 requires geographical separation of redundancy. Temporary external  
24 physical faults (transients) can lead to rapid redundancy attrition unless  
25 their effects can be undone. This means that it must be possible to recover  
26 corrupted processors. Temporary internal physical faults (intermittents) are  
27 treated as either permanent or transient faults according to their rate of  
28 recurrence.

29 Many design faults can also be tolerated like intermittents if their  
30 activation conditions are sufficiently diversified [Gray, 1986 #1234] (e.g.,  
31 through loosely-coupled replicated computations). However, design faults  
32 that are activated systematically for a given sequence of application inputs

1 can only be tolerated through diversification of design or specification. Due  
2 to limited resources, the project has not considered diversification of  
3 application software beyond imposing the requirement that no design  
4 decision should preclude that option in the future. However, we have  
5 studied the use of integrity level and control-flow monitoring mechanisms  
6 to ensure that design faults in non-critical application software do not affect  
7 critical applications. Moreover, we have considered diversification for  
8 tolerating design faults in off-the-shelf operating systems. We also  
9 encourage activation condition diversification so as to provide some  
10 tolerance of design faults in replicated hardware and replicated applications.

### 11 **2.1.3 Real-Time Models**

12 In keeping with the genericity objective, the architecture must be capable of  
13 supporting a range of real-time computational and scheduling models.

14 The *computational model* defines the form of concurrency (e.g.,  
15 tasks, threads, asynchronous communication, etc.) and any restriction that  
16 must be placed on application programs to facilitate their timing analysis  
17 (e.g., bounded recursion). Applications supported by GUARDS may  
18 conform to a time-triggered, event-triggered or mixed computational  
19 model.

20 Three *scheduling models* are considered [Wellings, 1998 #2163]:

- 21 • Cyclic — as typified by the traditional cyclic executive.
- 22 • Cooperative — where an application-defined scheduler and the  
23 prioritized application tasks explicitly pass control between themselves  
24 to perform the required dispatching.
- 25 • Pre-emptive — the standard pre-emptive priority scheme.

26 We have focused primarily on the pre-emptive scheduling model  
27 since this is the most flexible and the one that presents the greatest  
28 challenges.

## 1 2.2 The Generic Architecture

2 The diversity of end-user requirements and fault-tolerance strategies led us  
 3 to define a generic architecture that can be configured into a wide variety of  
 4 instances. The architecture favors the use of commercial off-the-shelf  
 5 (COTS) hardware and software components, with application-transparent  
 6 fault-tolerance implemented primarily by software. Drawing on experience  
 7 from systems such as SIFT [Melliari-Smith, 1982 #270], MAFT  
 8 [Kieckhafer, 1988 #766], FTPP [Harper, 1990 #1978] and Delta-4  
 9 [Powell, 1994 #1730], the generic architecture is defined along three  
 10 dimensions of fault containment (Figure 1) [Powell, 1997 #2012]:

- 11 • Integrity levels, or design-fault containment regions.
- 12 • Lanes, or *secondary* physical-fault containment regions.
- 13 • Channels, or *primary* physical-fault containment regions.

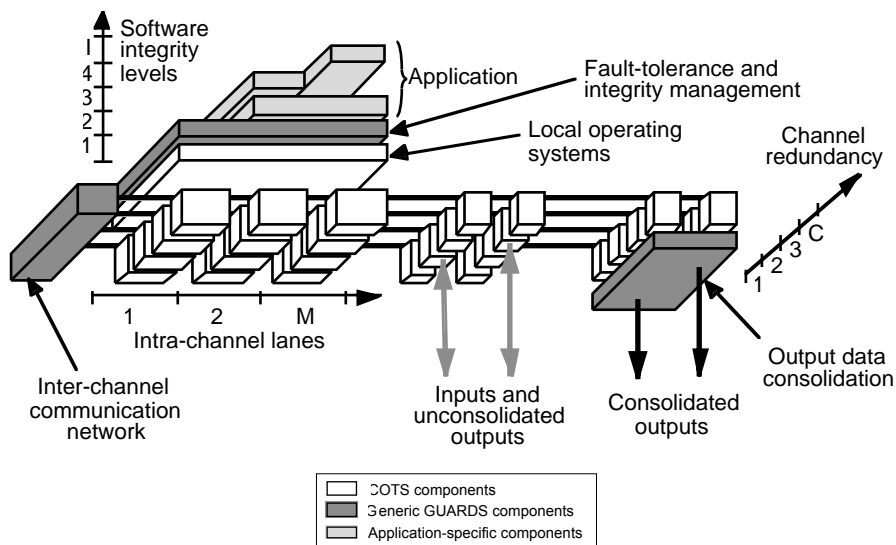


Figure 1 — The generic architecture

1           A particular instance of the architecture is defined by the dimensional  
2 parameters  $\{C, M, I\}$  and an appropriate selection of generic hardware and  
3 software GUARDS components. These generic components implement  
4 mechanisms for:

- 5     • Inter-channel communication.
- 6     • Output data consolidation.
- 7     • Fault-tolerance and integrity management.

8           Fault-tolerance and integrity management are software-implemented  
9 through a distributed set of generic system components (shown as a  
10 “middleware” layer on Figure 1). This layer is itself fault-tolerant (through  
11 replication and distribution of its components) with respect to faults that  
12 affect channels independently (e.g., physical faults). However, the tolerance  
13 of design faults in this system layer is not explicitly addressed<sup>1</sup>.

### 14   **2.2.1 The Integrity Dimension**

15       The integrity dimension aims to provide containment regions with respect  
16 to software design faults. The intent is to protect critical components from  
17 the propagation of errors due to residual design faults in less-critical  
18 components. Each application object is classified within a particular  
19 integrity level according to how much it can be trusted (the more  
20 trustworthy an object is, the higher its integrity level). The degree to which  
21 an object can be trusted depends on the evidence that is available supporting  
22 its correctness, and the consequences of its failure (i.e., its criticality).

23       The required protection is achieved by enforcing an integrity policy to  
24 mediate the communication between objects of different levels. Basically,  
25 the integrity policy seeks to prohibit flows of information from low to high  
26 integrity levels, like in the Biba policy [Biba, 1977 #1673]. However, this

---

<sup>1</sup> Note, however, that correlated faults are included in the models used to assess the dependability of instances of the architecture: see Section 2.8.2.

1 approach is inflexible. An object can obtain data of higher integrity than  
2 itself, but the data must then inherit the level of integrity of this object. This  
3 results in a decrease in the integrity of the data, without any possibility of  
4 restoring it. We deal with this drawback by providing special objects  
5 (*Validation Objects*) whose role is to apply fault tolerance mechanisms on  
6 information flows. The purpose of these objects is to output reliable  
7 information by using possibly corrupted data as input (i.e., with a low  
8 integrity level). Such objects upgrade the trustworthiness of data and hence  
9 allow information flows from low to high integrity levels [Totel, 1998  
10 #2151].

11 It must be ensured that it is not possible to by-pass the controls put  
12 into place to enforce the policy. This is achieved by spatial and temporal  
13 isolation, which are provided respectively by memory management  
14 hardware and resource utilization budget timers [Totel, 1998 #2165].  
15 Furthermore, for the most critical components (the topmost integrity level)  
16 and a core set of basic components (i.e., the integrity management  
17 components and the underlying hardware and operating systems), it must  
18 be assumed either that there are no design faults, or that they can be  
19 tolerated by some other means (e.g., through diversification).

20 \*\*\*

21 The integrity dimension of the architecture is detailed in Chapter  
22 \$\$\$7.

### 23 **2.2.2 The Lane Dimension**

24 Multiple processors or *lanes* are used essentially to define secondary  
25 physical fault containment regions. Such secondary regions can be used to  
26 improve the capabilities for fault diagnosis within a channel, e.g., by  
27 comparison of computation replicated on several nodes. There is also scope  
28 for improving coverage with respect to design faults by using intra-channel  
29 diversification.

30 Alternatively, lanes can be used to improve the availability of a  
31 channel, e.g., by passivating a node that is diagnosed to be permanently

1 faulty. The required fault diagnosis could be triggered either by the error-  
2 processing mechanisms within a channel or through an error signal from  
3 the inter-channel voting mechanisms.

4 Further reasons for defining an instance with multiple lanes include  
5 parallel processing to improve performance and isolation of software of  
6 different integrity levels. To aid the timing analysis of such software we  
7 require that the multiple processors within a channel have access to shared  
8 memory (see Section 2.5).

### 9 **2.2.3 The Channel Dimension**

10 Channels provide the *primary* fault containment regions that are the  
11 ultimate line of defense within a single instance for physical faults that  
12 affect a single channel. Fault tolerance is based on active replication of  
13 application tasks over the set of channels. It must be ensured that replicas  
14 are supplied with the same inputs in the same order, despite the occurrence  
15 of faults. Then, as long as replicas on fault-free channels behave  
16 deterministically, they should produce the same outputs. Error processing  
17 can thus be based on comparison or voting of replica outputs.

18 Not all instances require the same number of channels. In fact, one  
19 could imagine an instance with just one channel. This would be the case for  
20 an application that only requires multiple integrity levels, or for which the  
21 fault-tolerance mechanisms implemented within a channel are judged to be  
22 sufficient. It should be expected, however, that most applications require  
23 instances with several channels. Important cases are:

- 24 • Two channels: motivated either by a requirement for improved safety  
25 (using inter-channel comparison) or improved reliability (based on  
26 intra-channel self-checking to provide crash failure semantics).

- 1 • Three channels: the well-known triple modular redundancy (TMR)  
2 strategy that enables most<sup>2</sup> faults in one channel to be masked. In  
3 addition any disagreements are detected and used as inputs for error  
4 diagnosis and fault treatment.
- 5 • Four channels: to enable masking of completely arbitrary faults or to  
6 allow a channel to be isolated for off-line testing while still guaranteeing  
7 TMR operation with the remaining on-line channels.
- 8 Instances of the architecture with more than four channels are not  
9 currently envisaged.

### 10 **2.3 Inter-Channel Communication Network**

11 Central to the architecture is an inter-channel communication network  
12 (ICN), which fulfills two essential functions:

- 13 • It provides a global clock to all channels.
- 14 • It allows channels to achieve interactive consistency (consensus) on  
15 non-replicated data.

16 The ICN consists of an ICN-manager for each channel and  
17 unidirectional serial links to interconnect the ICN-managers. In the  
18 prototype implementation, the ICN-manager is a Motorola 68040-based  
19 board with a dual-port shared memory for asynchronous communication  
20 with the intra-channel VME back-plane bus. Serial links are provided by  
21 two Motorola 68360-based piggy-back boards<sup>3</sup>. Each such board provides  
22 two Ethernet links. One link is configured as transmit only, the other links  
23 are configured as receive only. An ICN-manager can thus simultaneously  
24 broadcast data to the remote ICN-managers over its outgoing serial link  
25 and receive data from the remote ICN-managers over the other links.

---

<sup>2</sup> The exception is that of Byzantine clock behavior (see Section 2.3.1).

<sup>3</sup> For a two-channel instance, only one ICN piggy-back board is necessary.

### 1 2.3.1 Clock Synchronization

2 The ICN-managers constitute a set of fully interconnected nodes. Each  
3 node has a physical clock and computes a global logical clock time through  
4 a fault-tolerant synchronization algorithm. Such an algorithm is classically  
5 defined as one that satisfies both the agreement and accuracy properties:

- 6 • The *agreement* condition is satisfied if and only if the skew between any  
7 non-faulty logical clocks is bounded.
- 8 • The *accuracy* condition is satisfied if and only if all non-faulty logical  
9 clocks have a bounded drift with respect to real time.

10 Since COTS-based solutions are preferred within GUARDS, we  
11 focused on software-implemented algorithms. In particular, we considered  
12 both convergence averaging and convergence non-averaging algorithms  
13 [Ramanathan, 1990 #1017].

14 In a convergence averaging algorithm, each node resynchronizes  
15 according to clock values obtained through periodic one-round clock  
16 exchanges. On each node, the other clocks can be taken into account  
17 through a mean-like function [Lamport, 1985 #239], or a median-like  
18 function [Lundelius-Welch, 1988 #1020]. The worst-case skew of these  
19 algorithms is dominated by the uncertainty on transmission delay. They  
20 can tolerate  $f$  arbitrarily faulty nodes in a (fully connected) network of  $n$   
21 nodes, under the sufficient condition that  $n > 3f$ .

22 In a convergence non-averaging algorithm, each node periodically  
23 seeks to be the system synchronizer. To deal with possible Byzantine  
24 behavior, the exchanged messages can be authenticated [Srikanth, 1987  
25 #353]. The worst-case skew of these algorithms is dominated by the  
26 maximum message transit delay. When authentication is used for inter-  
27 node message exchanges, they can tolerate  $f$  arbitrarily faulty nodes with  
28 only  $n > 2f$  nodes.

29 The GUARDS architecture uses a convergence-averaging solution  
30 based on [Lundelius-Welch, 1988 #1020] and applied to up to four nodes  
31 (i.e., ICN-managers in our architecture). This choice was motivated mainly

1 by reasons of performance and design simplicity. It implies that the  
2 probability of occurrence of a Byzantine clock must be carefully evaluated  
3 in a three-channel configuration. This probability is expected to be very  
4 small, since the ICN serial links are broadcast media and the ICN-  
5 managers can check that they receive a syntactically correct synchronization  
6 message in a well-defined local time window.

7 The global clock maintained by the set of ICN-managers is  
8 broadcasted, via the intra-channel back-plane busses, to the processors and  
9 I/O boards local to a channel.

### 10 **2.3.2 Interactive Consistency**

11 The issue of exchanging private data between channels and agreeing on a  
12 common value in the presence of arbitrary faults is known as the interactive  
13 consistency problem (the symmetric form of the Byzantine agreement  
14 problem) [Pease, 1980 #1937]. The two fundamental properties that a  
15 communication algorithm must fulfill to ensure interactive consistency are:

- 16 • *Agreement*: if channels  $p$  and  $q$  are non-faulty, then they agree on the  
17 value ascribed to any other channel.
- 18 • *Validity*: if channels  $p$  and  $q$  are non-faulty, then the value ascribed to  $p$   
19 by  $q$  is indeed  $p$ 's private value.

20 In the general case, the necessary conditions to achieve interactive  
21 consistency in spite of up to  $f$  arbitrarily faulty channels are [Lala, 1994  
22 #1884]:

- 23 • At least  $3f+1$  channels.
- 24 • At least  $2f+1$  disjoint inter-channel communication links.
- 25 • At least  $f+1$  rounds of message exchange.
- 26 • Bounded skew between non-faulty channels.

1 Under the assumption of authenticated messages, which can be  
2 copied and forwarded but not undetectably altered by a relay, the  
3 condition on the minimal number of channels can be relaxed to  $f+2$ .  
4 Nevertheless, at least  $2f+1$  channels are still necessary if majority voting  
5 must be carried out between replicated application tasks.

6 The interactive consistency protocol used in GUARDS is based on  
7 the ZA algorithm [Gong, 1998 #1992], which was derived from the Z  
8 algorithm [Thambidurai, 1988 #1993] by adding the assumption of  
9 authentication. In particular, authentication precludes the design fault in the  
10 Z algorithm identified in [Lincoln, 1993 #1444]. Following the hybrid fault  
11 model described in [Lincoln, 1993 #1444], the protocol allows for both  
12 arbitrarily faulty channels and channels affected by less severe kinds of  
13 faults (e.g., omission faults).

14 For performance reasons, and since by assumption the architecture  
15 only needs to tolerate accidental faults and not malicious attacks, we  
16 preferred to use a keyed checksum scheme for message authentication  
17 rather than resorting to true cryptographic signatures. Under this scheme,  
18 multiple checksums are appended to each (broadcast) message. Each  
19 checksum is computed over the concatenation of the data part of the  
20 message and a private key that is known only to the sender and to one of  
21 the broadcast destinations.

### 22 **2.3.3 Scheduling**

23 The ICN is scheduled according to a table-driven protocol. The schedule  
24 consists of a *frame* (corresponding to a given application mode) that is  
25 subdivided into *cycles* and *slots*. The last slot of a cycle is used for clock  
26 synchronization, so the length of a cycle is fixed either by the required  
27 channel synchronization accuracy or by the maximum I/O frequency in a  
28 given mode. The other slots of a cycle are of fixed duration and can support  
29 one fixed-sized message transmission (and up to three message  
30 receptions). In the current implementation, each message may contain 1000  
31 bytes.

1 \*\*\*

2 Further details on the inter-channel communication network and its  
3 implementation are given in Chapter \$\$\$3.

#### 4 **2.4 Inter-Channel Error Processing and Fault Treatment**

5 From a conceptual viewpoint, it is common to consider fault  
6 tolerance as being achieved by error processing and fault treatment  
7 [Anderson, 1981 #29].

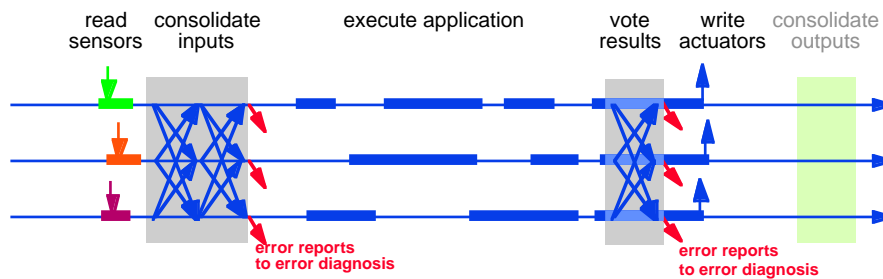
8 Error processing is aimed at removing errors from the computation  
9 state, if possible before failure occurrence. In general, error processing  
10 involves three primitives [Laprie, 1992 #982]: error detection, error  
11 diagnosis and error recovery.

12 Fault treatment is aimed at preventing faults from being activated  
13 again and also involves three primitives [Laprie, 1992 #982]: fault  
14 diagnosis, fault passivation and reconfiguration.

15 In GUARDS, error recovery is achieved primarily by error  
16 compensation, whereby the erroneous state contains enough redundancy to  
17 enable its transformation into an error-free state. This redundancy is  
18 provided by active replication of critical applications (although  
19 diversification is not precluded) over the  $C$  channels; it is application-  
20 transparent redundancy and managed by software, including comparison or  
21 voting of computed results. Error processing thus relies primarily on  $N$ -  
22 modular redundancy to detect disagreeing channels and (when  $C \geq 3$ ) to  
23 mask errors occurring in the voted results at run-time. When  $C=2$ , two  
24 possibilities are offered, as already mentioned in Section 2.2.3:

- 25 • Error detection (locally by a channel) and compensation (by switching  
26 to a single channel configuration).
- 27 • Error detection (by channel comparison) and switching to a safe state (a  
28 degenerate form of forward recovery).

1        Figure 2 illustrates the replicated execution of an iterative task in the  
 2 case of a three-channel configuration. After reading the replicated sensors,  
 3 the input values are consolidated across all channels after a two-round  
 4 interactive consistency exchange over the ICN. The application tasks are  
 5 then executed asynchronously, with pre-emptive priority scheduling  
 6 allowing different interleavings of their executions on each channel. This  
 7 diversifies the activities of the different channels, thereby allowing many  
 8 residual design faults to be tolerated as if they were intermittents (cf.  
 9 Section 2.1.2).



10

11

12

**Figure 2 — TMR execution of an application function split into sequential threads**

13

14

15

16

17

18

Application *state variables* (which contain values that are carried over between iterations) are used together with consolidated inputs to compute the output values, which are exchanged in a single round over the ICN and voted. The voted results are then written to the actuators, possibly via output consolidation hardware (cf. Figure 1 and Chapter \$\$\$6), which allows the physical values to be voted.

19

20

21

22

23

24

25

26

Since neither the internal state variables of the underlying COTS operating systems nor the totality of the application state variables are voted, further error recovery is necessary to correct any such state that becomes erroneous (note that this may be case even in the event of a transient fault). However, this is a secondary, non-urgent error recovery activity since, until another channel is affected by a fault, the error compensation provided by output voting or switching can be relied upon to ensure that correct outputs are delivered to the controlled process.

1 Consequently, this secondary error recovery can be viewed as part of fault  
2 treatment.

3 In the next section, we describe the GUARDS diagnosis  
4 mechanisms, which include both error diagnosis, to decide whether the  
5 damage to a channel's state warrants further action, and fault diagnosis to  
6 decide the location and type of the fault and thus the necessary corrective  
7 action.

8 Then, in Section 2.4.2, we describe the state recovery procedure that  
9 allows re-integration of a channel after a transient fault or repair of a  
10 permanent fault.

### 11 **2.4.1 Diagnosis**

12 The first step in diagnosis is to collect error reports generated during the  
13 interactive consistency and consolidation exchanges (majority voting  
14 discrepancies, timing errors, ICN bus transmission errors, protocol  
15 violations, etc.) and then to filter them to assess whether the extent of  
16 damage warrants further action. Indeed, some reported errors may not have  
17 resulted in any change to the state of a channel. Alternatively, if only a  
18 small part of the state has become erroneous, then an erroneous channel  
19 might correct itself autonomously by overwriting the erroneous variables  
20 during continued execution. If such fortuitous recovery does not occur, an  
21 explicit forward recovery action is necessary to reconstruct a correct state.

22 The filtering of errors is done using a software-implemented  
23 mechanism known as an  $\alpha$ -count, which was originally proposed for the  
24 discrimination of transient versus intermittent-permanent faults  
25 [Bondavalli, 1997 #2094]. Error reports are processed on a periodic basis,  
26 giving lower weights to error reports as they get older. A score variable  $\alpha_x$   
27 (initially set to 0) is associated to each component  $x$  to record information  
28 about the errors attributed to that component. The  $L$ -th judgement is  
29 accounted for as follows:

30  $\alpha_x(L) = \alpha_x(L-1) + 1$  if component  $x$  is perceived as faulty

1  $\alpha_x(L) = k \bullet \alpha_x(L-1)$  if component  $x$  is perceived as correct (with  $0 < k < 1$ )

2 When  $\alpha_x(L)$  becomes greater than a given threshold  $\alpha_T$ , the damage  
3 to the state of component  $x$  is judged to be such that further diagnosis is  
4 necessary.

5 The appropriate filtering action can be provided by several different  
6 heuristics for the accumulation and decay processes (where  $\alpha_x(L)$  takes  
7 slightly different expressions) [Bondavalli, 1997 #2094, Powell, 1998  
8 #2181, Rabéjac, 1997 #2153]. For a given error distribution, the  
9 parameters of the heuristics can be determined through a dependability  
10 evaluation (for example, see [Bondavalli, 1997 #2094]).

11 A distributed version of  $\alpha$ -count is used in GUARDS to provide the  
12 error syndrome that is input to inter-channel fault diagnosis. Each channel  $i$   
13 maintains  $C$   $\alpha$ -count variables, one,  $\alpha_{ii}$ , representing its opinion of its  
14 own health and  $C-1$  variables,  $\alpha_{ij}$ ,  $j \neq i$ , representing its opinions of the  
15 health of the other channels. The  $\alpha$ -counts are updated and processed  
16 cyclically. Each cycle  $N_\alpha$ , called an  $\alpha$ -cycle, has a duration chosen such  
17 that  $N_\alpha \cdot n_1 = N_{frame}$  where  $n_1$  is an integer and  $N_{frame}$  the duration of the  
18 ICN frame (see Section 2.3.3).

19 Since each channel may have a different perception of the errors  
20 created by other channels, the  $\alpha$ -counts maintained by each channel must  
21 be viewed as single-source (private) values. They are consolidated at the  
22 end of each  $\alpha$ -cycle through an interactive consistency protocol so that  
23 fault-free channels have a consistent view of the status of the instance (a  
24 consistent matrix  $A$  of  $\alpha$ -count values). During the next  $\alpha$ -cycle, fault  
25 diagnosis can thus be performed using  $A$ . The resulting diagnosis consists  
26 of a vector  $D$  whose elements  $D_i$  represent the diagnosed state of each  
27 channel (correct or requiring passivation and isolation).

1           The fault diagnosis problem has been extensively studied in the  
2 literature. An ideal diagnosis should be both *correct* and *complete*:

3     • A diagnosis is *correct* if any channel that is diagnosed as faulty is  
4       indeed faulty.

5     • A diagnosis is *complete* if all faulty channels are diagnosed as faulty.

6           In the current case, the inter-channel tests have imperfect coverage so  
7 a channel requiring passivation is not necessarily accused by all correct  
8 channels [Blough, 1992 #2155, Lee, 1990 #1509]. The algorithm in the  
9 current implementation diagnoses a channel as faulty if it is accused of  
10 being faulty by a majority of channels or, of course, if it accuses itself. This  
11 algorithm is correct and complete under the assumption that no more than  
12 one channel at a time is accused by a fault-free channel. However, due to  
13 the memory effect of the  $\alpha$ -count mechanism, this assumption can be  
14 violated if near-coincident faults occur on different channels. In this  
15 situation, there is thus a trade-off between the probability of incorrect  
16 diagnosis caused by a long memory effect (high value of  $k$ ) and the  
17 probability of having an incorrect majority vote due to slow elimination of  
18 a faulty channel (low value of  $k$ ). This trade-off is the subject of ongoing  
19 research.

20           Once a channel has been diagnosed as requiring passivation, it is  
21 isolated (i.e., disconnected from the outside world) and reset (with the re-  
22 initialization of operating system structures). A thorough self-test is then  
23 carried out. If the test reveals a permanent fault, the channel is switched off  
24 and (possibly) undergoes repair. Whenever a channel passes the test (i.e.,  
25 the fault was transient), or after having repaired a channel having suffered a  
26 permanent fault, it must be reintegrated to avoid unnecessary redundancy  
27 attrition.

28           It should be noted that the error filtering action of the  $\alpha$ -count can  
29 effectively be turned off by setting its threshold  $\alpha_T=1$ . In this case, any  
30 transient fault leading to a self-detected error (or to errors perceived by a  
31 majority of channels) will cause that channel to go through the possibly

1 lengthy self-test and re-integration procedure, irrespectively of the extent of  
2 the actual damage to the channel's state. A fault affecting another channel  
3 before re-integration of the former will induce a further decrease in the  
4 number of active channels. When transients are common, this policy can  
5 can cause rapid switching to a safe state if the number of active channels  
6 becomes insufficient for error compensation to remain effective. The  
7 choice of whether filtering is used or, more generally, the value of the  $\alpha$ -  
8 count threshold, thus leads to a classic trade-off between safety and  
9 reliability.

#### 10 **2.4.2 State Restoration**

11 For a channel to be reintegrated, it must first resynchronise its clock, then  
12 its state, with the pool of active channels. Since not all state variables are  
13 necessarily consolidated through ICN exchanges, the state (or *channel*  
14 *context*) cannot be retrieved by simply observing the traffic on the ICN, but  
15 must be explicitly copied from the active channels. This is achieved by a  
16 system state restoration (SR) procedure, called *Running SR*, applied to the  
17 channel context, i.e., the set of application state variables whose values are  
18 carried over successive iterations without consolidation.

19 A minimum level of service must be ensured, even during the SR  
20 procedure, so a limited number of vital application tasks must be allowed  
21 to continue execution on the active channels. Running SR is therefore a  
22 multi-step algorithm where, at each step, only a fraction of the state is  
23 exchanged. Furthermore, vital application tasks may update state variables  
24 while SR progresses.

25 The basic behavior of Running SR is the following (more details  
26 with variations and optimizations are given in [Bondavalli, 1997 #2106,  
27 Bondavalli, 1998 #2159]). The channel context is arranged in a single  
28 (logical) memory block managed by a "context object". When the state of  
29 channel needs to be restored, the system enters an "*SR mode*". The C-1  
30 active channels enter a "*put state*" sub-mode while the joining channel  
31 enters a "*get state*" sub-mode.



## 1 **2.5 Output Data Consolidation**

2       The purpose of the output data consolidation system (cf. Figure 1) is  
3 to map the replicated *logical outputs* of each channel onto the actual  
4 *physical outputs* to the controlled process, in such a way that the latter are  
5 either error-free or in a safe position. Such consolidation, placed at the  
6 physical interface with the controlled process, is the ultimate error  
7 confinement barrier, and is a complement to any software-implemented  
8 voting of the logical outputs.

9       A given instance of the architecture could have several different  
10 output consolidation mechanisms according to its various interfaces with  
11 the controlled process. Ideally, an output data consolidation mechanism  
12 should extend into the controlled process itself, to prevent the physical  
13 interface to the process from becoming a single point of failure. A typical  
14 example would be a control surface (e.g., in a fly-by-wire application) that  
15 can act as a physical voter by summing the forces produced by redundant  
16 actuators. Alternatively, a single channel can be designated to control a  
17 given actuator. Failures of that actuator can be detected at the application  
18 level by means of additional sensors allowing each channel to read back the  
19 controlled process variable and check it against the requested output.  
20 Recovery can then be achieved by switching to an alternative actuator.  
21 Other process-specific output data consolidation mechanisms used in the  
22 GUARDS end-user application domains include combinatorial logic  
23 implemented by relay or fluid valve networks, and the “arm-and-fire”  
24 technique commonly used to trigger space vehicle pyrotechnics (one  
25 channel sends an “arm” command, which is checked by the other channels,  
26 then all channels send matching “fire” commands; the pyrotechnics are  
27 triggered only if a majority of the latter concur with the former).

28       Output consolidation mechanisms such as these may be used for  
29 various end-user instances of the architecture. By definition, such process-  
30 specific techniques cannot be generic so no specific research has been  
31 carried out in this direction. However, the project has considered generic  
32 output consolidation mechanisms for networked and discrete outputs. A  
33 prototype consolidation mechanism is being implemented for discrete

1 digital or analogue outputs that can be electrically isolated from each other  
2 and then connected through a wired-OR to the output devices.  
3 Consolidation is achieved by having each channel read back its own output  
4 and those of the other channel(s) so that a vote can be carried out in  
5 software. Each channel then sends selection signals to a hard-wired voter  
6 (one per channel) that allows or disconnects that channel's outputs. This  
7 approach relies on the assumption that the output devices can tolerate the  
8 short but inevitable output glitch caused by the read-back, vote and  
9 disconnection delay.

10 \*\*\*

11 Chapter §§§6 gives further details regarding output consolidation.

## 12 **2.6 Real-Time Scheduling**

13 The architecture is capable of supporting a range of scheduling models (cf.  
14 Section 2.1.3). In this section, we focus on the standard pre-emptive  
15 priority-based scheme. We also discuss the consequences on scheduling of  
16 the ICN network.

17 Our timing analysis is based upon the Response-time Analysis  
18 [Audsley, 1993 #2029, Leung, 1982 #2164]. We assume that any  
19 communication between applications is asynchronous through the shared  
20 memory. The use of round-robin scheduling on the intra-channel VME  
21 bus allows all shared memory accesses to be bounded. This is adequate  
22 because it is assumed that the number of hosts within a channel is small.  
23 Furthermore, we assume the use of a non-blocking algorithm such as that  
24 proposed in [Simpson, 1990 #2177] to avoid the problems associated with  
25 remote blocking.

### 26 **2.6.1 Inter-Channel Replication of Applications**

27 For an application task to be replicated, it must behave deterministically and  
28 each replica task must process the same inputs in the same order. At any  
29 point where there is potential for replica divergence, the channels must

1 perform an interactive consistency agreement. Unfortunately, the cost of  
 2 executing interactive consistency agreement protocols can be significant.  
 3 There is therefore a need to keep their use to a minimum.

4 In our approach, we force all replicated tasks to read the same  
 5 internal data. We can thus trade-off fewer agreement communications (and  
 6 therefore greater efficiency) against early detection of errors. If we assume  
 7 that each replica does not contain any inherently non-deterministic code,  
 8 replica determinism and error masking (or detection) can be ensured by:

- 9 • Performing interactive consistency agreement or Byzantine agreement  
 10 on single-sourced data.
- 11 • Ensuring that all replicas receive the same inputs when those inputs are  
 12 obtained from other replica tasks (replicated inputs).
- 13 • Voting on any vital output.

#### 14 **2.6.1.1 Agreement on Sensor Inputs**

15 To reduce the complexity of the input selection algorithm, which processes  
 16 the vector of redundant values consolidated through the interactive  
 17 consistency exchange, it is important to minimise the error between the  
 18 redundant input values. However, since the tasks are independently  
 19 scheduled on each channel, they could read their corresponding sensors at  
 20 significantly different times. This is similar to the input jitter problem  
 21 where a task ( $\tau$ ) implementing a control law has to read its input on a  
 22 regular basis. If jitter is a problem, the solution is to split the task into two  
 23 tasks ( $\tau^{\text{ip}}, \tau'$ ).  $\tau^{\text{ip}}$  has a release time<sup>4</sup> and a deadline appropriate for the  
 24 dynamics (and the allowable jitter) of the physical quantity being measured  
 25 by the sensor. Task  $\tau'$  has the original  $\tau$ 's deadline and is executed at an

---

<sup>4</sup> We assume that all I/O is periodic in nature.

1 offset from the release time of  $\tau^p$ . We will discuss what value this offset  
2 should have in Section 2.6.2).

### 3 **2.6.1.2 Identical Internal Replicated Input**

4 Two cases need to be considered when reader and writer tasks share the  
5 same data, according to whether or not there is an explicit precedence  
6 constraint between the writer and the reader. When there is such a  
7 constraint, then it can be captured by the scheduling. When tasks share data  
8 asynchronously (and therefore there is no explicit precedence constraint  
9 between the writer and the reader), there are four types of interaction:

- 10 • Periodic writer — Periodic reader: the periods of the two tasks do not  
11 have a simple relationship.
- 12 • Periodic writer — Sporadic reader: there is no relationship between the  
13 period of the writer and the release of the reader.
- 14 • Sporadic writer — Sporadic reader: there is no relationship between the  
15 release of the writer and the release of the reader.
- 16 • Sporadic writer — Periodic reader: there is no relationship between the  
17 release of the writer and the period of the reader.

18 In all of these cases, to ensure each replica reads the same value, we  
19 keep more than one copy of the data (usually two is enough) and use  
20 timestamps [Barrett, 1995 #2107, Poledna, 1998 #2027]. The essence of  
21 this approach is to use off-line schedulability analysis [Audsley, 1993  
22 #2029] to calculate the worst-case response times of each replicated writer.  
23 The maximum of these values is added to the release time of the replicas  
24 (taking into account any release jitter) to give a time by which all replicas  
25 must have written the data (in the worst case). To allow for clock drift  
26 between replicas, the maximum skew,  $\epsilon$ , is also added. This value is used  
27 as a timestamp when the data is written.

1        A reader replica simply compares its release time with the data  
2 timestamp. If the timestamp is earlier, then the reader can take the data. If  
3 the timestamp is later than its release time, then the reader knows that its  
4 replicated writer has potentially executed before the other replicated writers.  
5 It must therefore take a previous value of the data (the most recent) whose  
6 timestamp is earlier than its release time. All reader replicas undertake the  
7 same algorithm and consequently get the same value.

### 8    **2.6.1.3 Output Voting**

9        Where output voting is required, it is again necessary to transform the  
10 replicated task writing to the actuators into two tasks ( $\tau'$  and  $\tau^{op}$ ):  $\tau'$  sends  
11 the output value across the ICN for voting, and  $\tau^{op}$  reads the majority vote  
12 and sends this to the actuator. The deadline of  $\tau'$  will determine the earliest  
13 point when the ICN manager can perform the voting. The offset and  
14 deadline of  $\tau^{op}$  will determine when the voted result must be available and  
15 the amount of potential output jitter. Hence, the two tasks have similar  
16 timing characteristics to the tasks used for input agreement (Section  
17 2.6.1.1). The main difference is that there is a simple majority vote rather  
18 than an agreement protocol involving three separate values.

### 19    **2.6.2 Handling Offsets**

20        A real-time periodic transaction model has been developed in which  
21 periodic transaction  $i$  consists of three tasks  $\tau_i^1$ ,  $\tau_i^2$  and  $\tau_i^3$ . Task  $\tau_i^1$  reads a  
22 sensor and sends the value to the ICN manager. Task  $\tau_i^2$  reads back from  
23 the ICN manager the set of values received from all the replicas; it  
24 consolidates the values and processes the consolidated reading and  
25 eventually produces some data. It sends this data for output result  
26 consolidation to the ICN manager. Task  $\tau_i^3$  reads the consolidated result  
27 from the ICN manager and sends it to the actuator.

1           This form of real-time transaction is implemented by timing offsets.  
2 Analysis of task sets with offsets is N-P complete [Leung, 1982 #2164]  
3 and even sub-optimal solutions are complex [Audsley, 1993 #2179,  
4 Audsley, 1993 #2182, Tindell, 1993 #2180]. The approach we take is  
5 based on [10], modified to take into account the fact that the computational  
6 times of  $\tau_i^1$  and  $\tau_i^3$  (respectively  $C_i^1$  and  $C_i^3$ ) are much smaller than  $C_i^2$ , the  
7 computational time of  $\tau_i^2$ , i.e.,  $C_i^2 \gg \max(C_i^1, C_i^3)$ .

8           Once offsets has been assigned, a check must be made to ensure that:  
9 1. the response times of the individual tasks are less than the offsets of the  
10 next task in the transaction,  
11 2. there is enough time before the offset and after the response to transmit  
12 data on the ICN network, and  
13 3. that the deadline of the transaction has been met.

14           If any of these conditions is violated, then it may be possible to  
15 modify the offsets of the transaction violating the condition in an attempt to  
16 satisfy all the requirements [Bates, 1997 #2160].

### 17 **2.6.3 Scheduling the ICN Network**

18 Following the individual schedulability analysis of each channel, the  
19 following characteristics are known for each task participating in replicated  
20 transactions:

- 21 • Period
- 22 • Response-time
- 23 • Offset
- 24 • Deadline

25           The ICN tables can be built from this information— in the same way  
26 as cyclic executive schedules can be constructed [Burns, 1995 #2178].

1 Since all communication through the channels' shared memory is  
2 asynchronous, the ICN manager can take the data any time after the  
3 producing task's deadline has expired.

4 Of course, there is a close relationship between the scheduling of the  
5 channels and the scheduling of the ICN network. If the off-line tool fails to  
6 find an ICN schedule, it is necessary to re-visit the design of the  
7 application.

8 \*\*\*

9 Further details regarding scheduling are given in Chapter \$\$\$4.

## 10 **2.7 Architecture Development Environment**

11 The generic architecture is supported by an Architecture Development  
12 Environment (ADE) [Paganone, 1997 #2169] consisting of a set of tools  
13 for designing instances of the architecture according to a coherent and  
14 rigorous design method. The tool-set allows collection of the performance  
15 attributes of the underlying execution environment and the analysis of the  
16 schedulability of hard real-time threads, not only within each processing  
17 element of the system, but also among them. This allows in particular a  
18 rigorous definition of critical communication and synchronization among  
19 the redundant computers.

### 20 **2.7.1 Design Method**

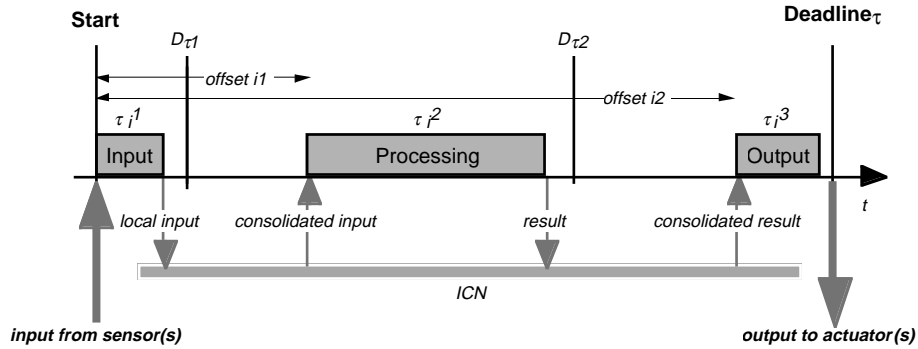
21 The design and development of a GUARDS software application are  
22 centered on a hard real-time (HRT) design method, which allows real-time  
23 requirements to be taken into account and verified during the design. The  
24 method also addresses the problem of designing replicated, fault-tolerant  
25 architectures, where a number of computing and communication boards  
26 interact for the consolidation of input values and output results.

27 The design of a GUARDS application is defined as a sequence of  
28 correlated activities, that may be re-iterated to produce a software design

- 1 that complies with both the functional and non-functional requirements of  
2 the application. Three design activities are identified:
- 3 • *Functional* architecture design, where the software application is defined  
4 through an appropriate design method and according to its functional  
5 requirements and its performance requirements (task periods, deadlines,  
6 etc.).
  - 7 • *Infrastructure* architecture design, where the required hardware boards  
8 and generic GUARDS software components are identified. They  
9 constitute the underlying computing environment of the application  
10 software.
  - 11 • *Physical* architecture design, where the functional architecture is  
12 mapped onto the infrastructure and analyzed according to the  
13 performance requirements. This is done not only for the processors  
14 within each replicated channel, but also at the inter-channel level, to  
15 determine the ICN exchanges needed to consolidate input values and  
16 output results.

### 17 **2.7.2 Inter-Channel Schedulability**

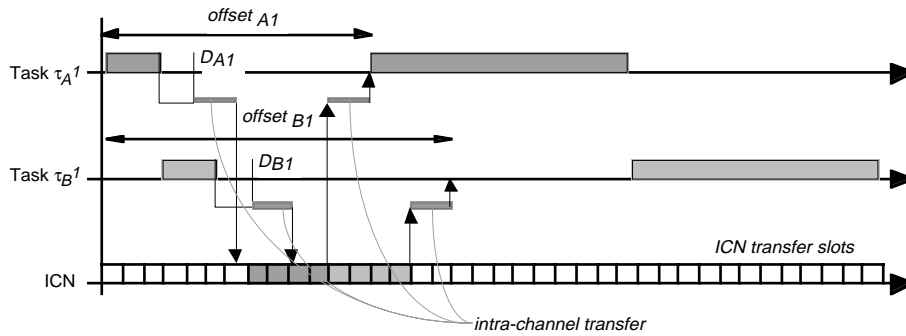
18 According to the dependability requirements, each critical application task  
19 replica needs to consolidate its inputs and its output results with those of  
20 the corresponding replicas on the other channels (Figure 3). Each  
21 application task  $\tau_i$  is structured as a real-time transaction, consisting of  
22 three sub-tasks, or threads,  $\tau_i^1$ ,  $\tau_i^2$  and  $\tau_i^3$  responsible for input acquisition,  
23 result calculation and output actuation (cf. Section 2.6.2).



1  
2

**Figure 3 — Decomposition of critical tasks**

3 For each application task  $\tau_i$ , a deadline is set that defines the time by  
 4 which the final value must be sent to the actuator(s) (corresponding to the  
 5 third thread of the transaction). Intermediate deadlines  $D_{\tau_1}$  and  $D_{\tau_2}$   
 6 are also introduced for the first and second threads. They define the time by  
 7 which the input or output results are (or must be) ready for transfer through  
 8 the ICN (after a fixed intra-channel transfer time) and consolidated. The  
 9 transfer and consolidation of each value over the ICN must take place at  
 10 pre-defined transfer slots (to synchronize such activities on each channel)  
 11 and the needed duration determines an offset for the activation of the  
 12 following thread (Figure 4).



13  
14

**Figure 4 — Inter-channel schedulability**

1           Although the final deadline is set by the requirements, intermediate  
2 deadlines can be set arbitrarily during the design, according to the intra-  
3 channel schedulability analysis and the allocation of ICN transfer slots.  
4 Different intermediate deadlines may imply different ICN transfers (i.e.,  
5 slot allocations) and consequently different offsets. Consequently, the HRT  
6 schedulability analysis (at the intra-channel level) must take into account the  
7 possible tuning of HRT design attributes (i.e., intermediate deadlines and  
8 offsets), as well as the slot allocation (i.e., the inter-channel schedulability).

### 9   **2.7.3 Supporting Tools**

10   The *functional* architecture design is supported by an appropriate method  
11 and tool. To provide for genericity, GUARDS does not force the selection  
12 of a specific method, but it is assumed that the method selected by the user  
13 is indeed suitable for the design of real-time software systems.  
14 Nevertheless, a survey and an analysis of design methods have shown that  
15 only HRT-HOOD [Burns, 1995 #2170] addresses explicitly the design of  
16 *hard* real-time systems, providing means for the verification of their  
17 performance. Therefore, HRT-HOOD was selected as the baseline design  
18 method and HRT-HoodNICE adopted as supporting tool [Intecs, 1996  
19 #2171].

20   However, the analysis also revealed several weaknesses of the  
21 method, in particular related to the design of *distributed* systems. The  
22 method was thus extended to include the concept of *Virtual Nodes*, similar  
23 to that in the HOOD 3.1 method [HOOD, 1992 #2172]. The extended  
24 method can take into account the lane dimension of GUARDS (by  
25 allocating objects to different processors within a channel) and the integrity  
26 dimension (by defining spatial firewalls around objects of a given  
27 criticality). The HRT-HoodNICE toolset has been accordingly enhanced.

28   The *infrastructure* architecture design is supported by a specific  
29 toolset that manages an archive of hardware and software components.  
30 Such components are described by their relations, compatibilities and  
31 performance attributes. The tool selects the needed components according  
32 to the characteristics of the required GUARDS instance.

1           As part of the *physical* architecture design, the application tasks (i.e.,  
2 HRT objects) identified in the functional architecture are mapped onto the  
3 infrastructure architecture. They are coupled with the real-time models of  
4 the selected components, in order to analyze and verify their schedulability  
5 properties. This is done by the Temporal Properties Analysis toolset, which  
6 analyzes the performance of the resulting distributed software system.

7           The Temporal Properties Analysis tool-set includes a Schedulability  
8 Analyzer and a Scheduler Simulator, based on those available in HRT-  
9 HoodNICE. They have been enhanced to provide a more precise and  
10 realistic analysis (by taking into account the concept of thread offsets) and  
11 to cope with the specific needs of a redundant fault-tolerant architecture (by  
12 allowing the analysis of the interactions over the ICN).

13           A further result of the physical architecture design is that, on the  
14 basis of the real-time models produced by the verification tools, the critical  
15 interactions among software functions on different channels are scheduled  
16 in a deterministic way. The ICN transfer slots allocated to them and a set of  
17 pre-defined exchange tables are produced automatically.

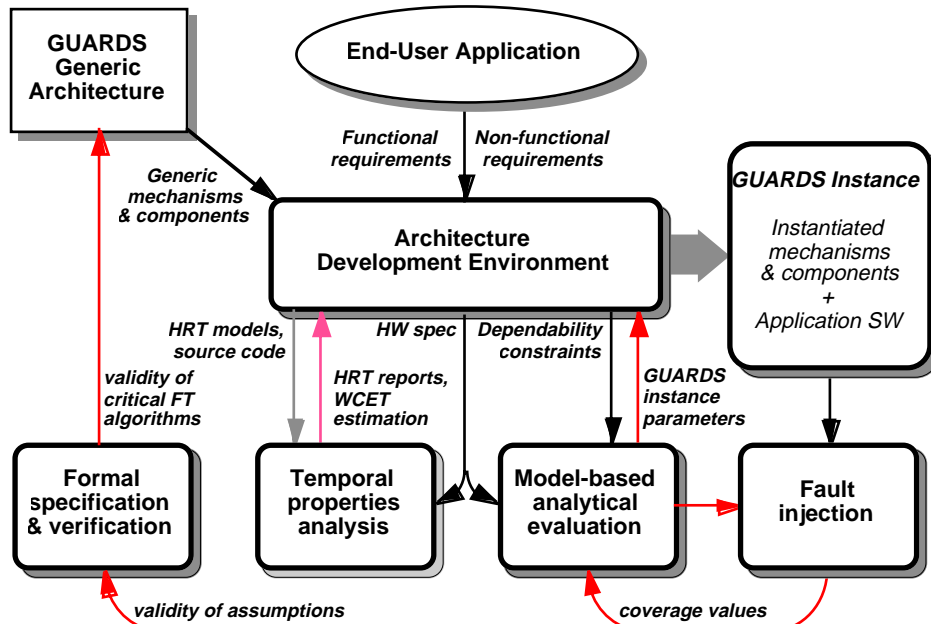
18           As a final step of the design phase, the overall structure of the  
19 software application is extracted from the HRT-HOOD design and the  
20 related code is automatically generated. To this end, a set of mapping rules  
21 has been defined to translate the HRT-HOOD design in terms of threads  
22 implemented in a sequential programming language (which could be C or  
23 the sequential subset of Ada) and executed by a POSIX compliant  
24 microkernel [Wellings, 1997 #2173].

## 25   **2.8 Validation**

26   The validation strategy implemented within GUARDS has two main  
27 objectives [Arlat, 1997 #2054]:

- 28   • A short-term objective: the validation of the design principles of the  
29    generic architecture, including both real-time and dependability  
30    mechanisms.

- 1     • A long-term objective: the validation of the development of instances of  
2       the architecture implementing specific end-user requirements.
- 3       A large spectrum of methods, techniques and tools has been  
4       considered to address these validation objectives and to account for the  
5       validation requirements expressed by the emerging trans-application  
6       domain standard IEC 1508 [IEC 1508, #2122].
- 7       Following the comprehensive development model described in  
8       [Laprie, 1995 #2140], the validation strategy is closely linked to the design  
9       solutions and the proposed generic architecture. The validation environment  
10      that supports the strategy includes components for verification and  
11      evaluation, using both analytical and experimental techniques. Figure 5  
12      illustrates the relationship between the components of the validation  
13      environment, and their interactions with the architecture development  
14      environment.



1  
2  
3

**Figure 5 — Main interactions between architecture development and validation**

4 Besides the three main validation components (namely, formal  
5 verification, model-based evaluation and fault injection), the figure  
6 explicitly identifies the role played by the methodology and the supporting  
7 tool-set being developed for schedulability analysis (cf. Section 2.7.3). The  
8 figure also depicts the complementarity and relationships among the three  
9 validation components. In particular, fault injection (carried out on  
10 prototypes) complements the other validation components by providing  
11 means for: a) assessing the validity of the necessary assumptions made by  
12 the formal verification task, and b) estimating the coverage parameters  
13 included in the analytical models for dependability evaluation. The  
14 following three subsections briefly describe the related validation activities.  
15 Further details are given in Chapter \$\$\$10.

### 1 2.8.1 Formal Verification

2 Formal approaches were used both for specification and as a design-aid.  
 3 We concentrated our effort on four dependability mechanisms, which  
 4 constitute the basic building blocks of the architecture: a) clock  
 5 synchronisation, b) interactive consistency, c) fault diagnosis, and d) multi-  
 6 level integrity.

7 The formal approaches that have been applied included both theorem  
 8 proving and model checking. Table 1 summarizes the main features of the  
 9 verifications carried out for each of the target mechanisms.

10 **Table 1 — Formal verification approaches**

Target Mechanism	Clock Synchronization	Interactive Consistency	Fault Diagnosis	Multi-level Integrity
Properties Verified	Agreement Accuracy	Agreement Validity	Correctness Completeness	Segregation Policy (Multi-level Objects)
Approach	Theorem Proving	Model Checking		
Description and Specification	Higher Order Logic	Process Algebra (CCS) and Temporal Logic (ACTL)		
Supporting Tool	PVS	JACK		

11 CCS: Calculus for Communicating Systems

12 ACTL: Action based version of Computation Tree Logic

13 The work carried out on the verification of clock synchronization  
 14 relied heavily on PVS (*Prototype Verification System*) [Owre, 1996  
 15 #2198]. It led to the development of a general theory for averaging and  
 16 non-averaging synchronization algorithms [Schwier, 1997 #2195]. The  
 17 verification of the synchronization solution used in GUARDS (cf.  
 18 Section 2.3.1) was derived as an instantiation of this general theory.

1           The verifications concerning interactive consistency [Bernardeschi,  
2 1998 #2184, Bernardeschi, 1998 #2185], fault diagnosis [Bernardeschi,  
3 1998 #2202] and multi-level integrity [Semini, 1998 #2196] were all based  
4 on model checking using the JACK (*Just Another Concurrency Kit*) toolset  
5 [Bouali, 1994 #2187]. This integrated environment provides a set of  
6 verification tools that can be used separately or in combination. Due to the  
7 complexity of the required models, the tool-set was extended to include a  
8 symbolic model checker for ACTL [Fantechi, 1998 #2189].

9           These studies demonstrated the feasibility and the benefits of formal  
10 methods on realistic industrial problems using state-of-the-art tools. We  
11 believe this is an important outcome that can significantly facilitate the  
12 acceptance of the GUARDS generic architecture for critical applications. It  
13 is also expected that further exploitation of the complementarity between  
14 theorem proving and model checking could facilitate a wider industrial  
15 acceptance of formal methods.

## 16 **2.8.2 Dependability Evaluation**

17 Model-based dependability evaluation is widely recognized as a powerful  
18 means to make early and objective design decisions by assessing  
19 alternative architectural solutions. Nevertheless, fault-tolerant distributed  
20 systems (such as GUARDS instances) pose several practical modeling  
21 problems (“stiffness”, combinatorial explosion, etc.). Moreover, due to the  
22 variety of the application domains being considered, the dependability  
23 measures of interest encompass reliability, availability and safety.

24           To cope with these difficulties, we adopted a divide-and-conquer  
25 approach, where the modelling details and levels are tailored to fit the needs  
26 of the specific evaluation objectives. This was achieved by focusing first  
27 the modelling effort either on generic or specific architectural features, or  
28 on selected dependability mechanisms. Then, an abstract modelling  
29 viewpoint that aims to provide a global framework for configuring  
30 instances to meet specific application dependability requirements was  
31 devised. Finally, elaborating on previous related work (e.g., [Kanoun, 1996

1 #2080]), a detailed modelling viewpoint that supports incremental and  
2 hierarchical evaluation has been considered.

3 Table 2 identifies the various dependability evaluation activities  
4 carried out according to these three modeling viewpoints.

5 **Table 2 — Dependability evaluation viewpoints and studies**

<b>Modeling Level</b>	<b>Focused</b>	<b>Abstract</b>	<b>Detailed</b>
<b>Targeted Mechanisms, Strategies, Instances</b>	<ul style="list-style-type: none"> <li>- <math>\alpha</math>-count mechanism</li> <li>- Phased missions</li> <li>- Intra-channel error detection mechanism</li> </ul>	<ul style="list-style-type: none"> <li>- Railway prototype Instance</li> <li>- Nuclear submarine prototype Instance</li> <li>- Space prototype instance</li> </ul>	Overall design and interactions (Nuclear submarine prototype instance)
<b>Formalism</b>	Stochastic activity networks and generalized stochastic Petri nets	Generalized stochastic Petri nets	Stochastic Petri nets
<b>Supporting Tools</b>	UltraSAN, SURF-2	SURF-2	MOCA-RP
<b>Resolution Methods</b>	Analytical and Monte-Carlo simulation	Analytical and method of stages	Monte-Carlo simulation

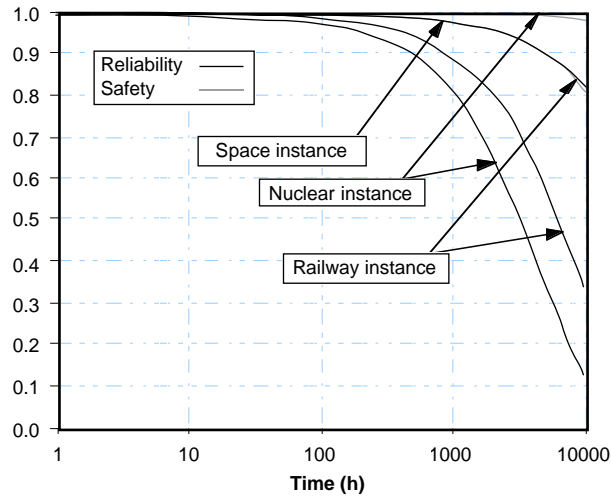
6 The focused models addressed several issues concerning the analysis  
7 of generic mechanisms (e.g.,  $\alpha$ -count [Bondavalli, 1997 #2094]) and of  
8 specific features for selected instances (phased missions, for the space  
9 prototype instance [Bondavalli, 1997 #2186], intra-channel error detection  
10 for the railway prototype instance).

11 The second viewpoint aims to establish a baseline set of models of  
12 the three prototype instances of the architecture [Powell, 1998 #2162]. A  
13 general notation is introduced that allows for a consistent interpretation of  
14 the model parameters (layers, correlated faults, etc.) for each prototype  
15 instance. This work provides the foundation of a generic modeling  
16 approach to guide the choice of a particular instantiation of the architecture,

1 according to the dependability requirements of the end-user application. A  
2 large number of parameters (proportion of transient vs. permanent faults,  
3 correlated faults in the hardware and software layers, coverage factors,  
4 error processing rates, etc.) have been included in the models, allowing  
5 intensive sensitivity analyses to be carried out. As an example of the results  
6 obtained, Figure 6 compares reliability and safety for the three instances  
7 considered<sup>5</sup>, for one set of values of the parameters included in the models.  
8 The ranking of the reliability curves simply reflects the redundancy at the  
9 channel level ( $C = 4, 3,$  and  $2$  for the space, railway and space instances,  
10 respectively). However, in the case of safety, the ranking of the nuclear and  
11 railway instances is reversed. This is mainly due to the fact that, in the  
12 nuclear instance, correlated design faults in either lane of the executive layer  
13 can be detected by the inter-lane comparison within each channel.

---

<sup>5</sup> As safety is only a minor concern for the application targeted by the space instance, only the reliability curve is shown for that instance.



1

Parameters	Railway	Nuclear	Space
<b>Failure rates (h<sup>-1</sup>)</b>			
<b>Hardware components</b>			
ICN manager (10 <sup>-5</sup> )			
- independent	.999 10 <sup>-5</sup>	.999 10 <sup>-5</sup>	.999 10 <sup>-5</sup>
- correlated (interchannel)	10 <sup>-8</sup>	10 <sup>-8</sup>	10 <sup>-8</sup>
Processor (10 <sup>-5</sup> )			
- independent	.99 10 <sup>-5</sup>	.989 10 <sup>-5</sup>	.989 10 <sup>-5</sup>
- correlated (intrachannel)	0	0	0
- correlated (interchannel)	10 <sup>-7</sup>	10 <sup>-7</sup>	10 <sup>-7</sup>
- correlated (global)	0	10 <sup>-8</sup>	10 <sup>-8</sup>
<b>Software layers:</b>			
correlated (interchannel) only			
Executive		10 <sup>-4</sup>	
System		10 <sup>-6</sup>	
Application (integrity lev. 1)		10 <sup>-5</sup>	
Application (integrity lev. 2)		10 <sup>-6</sup>	
<b>Coverage factors</b>			
ICN manager		.80	
Processor & software layers	.80 (Except, Nuclear .99)		
Inter-channel voting		1	
Integrity policy		.80	
Self-diagnosis		.99	
<b>Others</b>			
Proportion of transient faults		90 %	
Self-diagnosis rate		100 h <sup>-1</sup>	
Restoration rate		100 h <sup>-1</sup>	
Repair rate		.1 h <sup>-1</sup>	

2

Figure 6 — Comparison of the dependability of selected instances

1 Detailed models are needed to allow for a more comprehensive  
2 analysis of the behavior of the instances (dependencies, error propagation,  
3 etc.). Specific work has addressed hierarchical modeling with the aim of  
4 mastering the complexity of such detailed models [Jenn, 1998 #2191].  
5 This work is directed mainly at: a) enforcing the thoroughness of the  
6 analysis, b) helping the analyst (i.e., a design engineer who is not  
7 necessarily a modeling expert). It is currently being applied and refined on  
8 the nuclear submarine prototype instance.

9 Although they were supported by different tools, namely UltraSAN  
10 [Sanders, 1993 #1459], MOCA-RP [Dutuit, 1997 #2199] and SURF-2  
11 [Béounes, 1993 #1407], the modeling efforts all rely on the stochastic Petri  
12 net formalism. This should facilitate re-use of the results of the various  
13 studies (both models and modeling methodology).

### 14 **2.8.3 Fault Injection**

15 The main objectives for the planned fault injection activities are twofold:  
16 a) to complement the formal verification of GUARDS mechanisms (i.e.,  
17 removal of residual deficiencies in the mechanisms), and b) to support the  
18 development of GUARDS instances by assessing its overall behavior in  
19 the presence of faults, in particular by estimating coverage and latency  
20 figures for the built-in error detection mechanisms [Arlat, 1990 #34].

21 Indeed, as an experimental approach, fault injection provides a  
22 pragmatic means to complement formal verification by overcoming some  
23 of the behavioral and structural abstractions made, especially regarding the  
24 failure mode assumptions. Fault injection is to be carried out on complete  
25 prototypes so the mechanisms are tested globally when they have been  
26 integrated into an instance. In particular, the interactions between the  
27 hardware and software features are taken into account.

28 Although available tools could have been used — albeit with some  
29 extensions — a specific fault injection tool-set (FITS) is being developed.  
30 Such a tool-set is a major feature of the validation environment made

1 available to support the end-users in the development of specific instances  
2 of the generic architecture.

3 Both for cost-effectiveness and flexibility, the fault injection  
4 environment is based on the software-implemented fault injection (SWIFI)  
5 technique [Hsueh, 1997 #2100]. This also allows tests to be conducted  
6 more efficiently, since: a) a limited number of errors can simulate the  
7 consequences of a large number of faults, b) it is less likely that the injected  
8 error fails to exercise the dependability mechanisms.

9 Two main levels of injection are being considered, whether the  
10 targeted mechanisms are implemented by the ICN-manager board or by  
11 the intra-channel processors. In practice, the implementations differ  
12 significantly: whereas fault injection on the intra-channel processors can be  
13 assisted by the resident COTS operating systems and debug facilities  
14 [Carreira, 1998 #2188, Krishnamurthy, 1998 #2192], the ICN-manager  
15 only has a very simple cyclic executive.

16 We concentrate here on the verification objective, which is the main  
17 focus of the current implementation of FITS; further features are needed to  
18 address the evaluation objective (e.g., see [Arlat, 1993 #810]). Some  
19 examples of the experiments aimed at testing various mechanisms are  
20 given in Table 3.

1 **Table 3 — Fault injection-based testing of the GUARDS dependability**  
 2 **mechanisms**

	<b>Fault/Error Type</b>	<b>Trigger Event</b>	<b>Observation</b>
<b>ICN Mechanisms:</b> - Clock Synchronization - Interactive Consistency - $\alpha$ -count and Diagnosis - ...	- Omit/Delay Synch. Mess. - Alter ICN Table - Repeat the Same Error	Specific Frame/Cycle/Slot	ICN Status Vectors
<b>Host Mechanisms:</b> - Multi-level Integrity - Control Flow Monitoring - HRT Scheduling - ...	- Issue a Forbidden Call - Provoke Illegal Branch - Modify Task Priorities	Condition (Flag/Counter) on the Host	- Integrity Kernel Activity - Control Flow Monitor - ICN Status Vectors

3 Besides injecting specific fault/error types, FITS allows injection to  
 4 be synchronized with the target system by monitoring trigger events. Of  
 5 course, the observations depend on the targeted mechanisms. While it is  
 6 primarily intended to inject on a single channel, observations are carried out  
 7 on all channels. Initial experiments will focus on the ICN mechanisms.

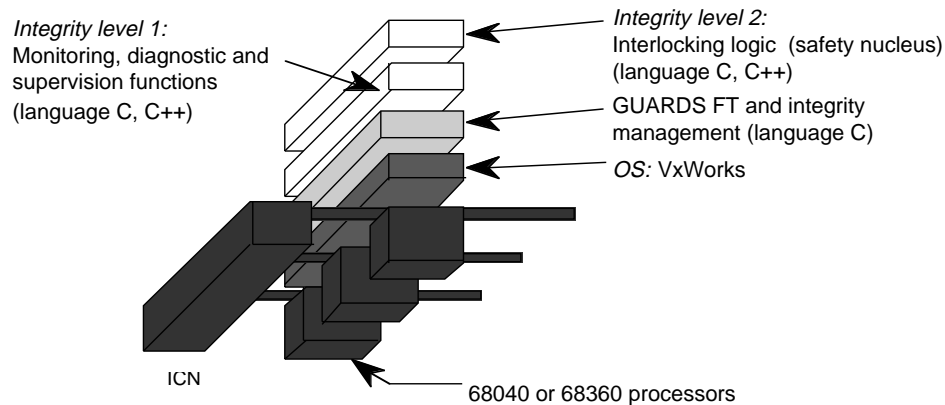
## 8 2.9 Prototypes

9 Several practical instances of the generic architecture have been  
 10 studied, and a prototype for each of the three end-user domains is under  
 11 development. The basic building blocks are practically identical in each  
 12 instance. However, the configurations of the instances are very different  
 13 and offer quite different fault-tolerance strategies. Moreover, although the  
 14 operating systems chosen by each end-user are POSIX-compliant, they are

1 not identical, neither are the end-users' preferred system development  
 2 environments. Consequently, although there is a single specification of the  
 3 generic software components of the fault-tolerant and integrity  
 4 management layer, they have different practical instantiations in each  
 5 instance.

### 6 2.9.1 Railway Instances

7 One instance studied for the railway domain is a fairly classic triple  
 8 modular redundant (TMR) architecture with one processor per channel  
 9 (Figure 7). If a channel is diagnosed to be permanently faulty, the system  
 10 degrades to a two-out-of-two mode. If a fault should occur while in this  
 11 mode, the instance is switched to a safe state if the errors caused by the  
 12 fault are detected (either locally within a channel or by two-out-of-two  
 13 comparison).



14

15

**Figure 7 — Railway triplex instance ( $C=3, M=1, I=2$ )**

16

17

18

This instance would employ Motorola 68040 or 68360 processors, each running a POSIX-compliant VxWorks operating system. Compared to currently deployed systems, the innovative aspect of this instance is the

1 co-existence of two levels of application software integrity corresponding  
2 to very different degrees of criticality:

3 • Highly-critical interlocking logic or safety nucleus, which must be the  
4 highest integrity.

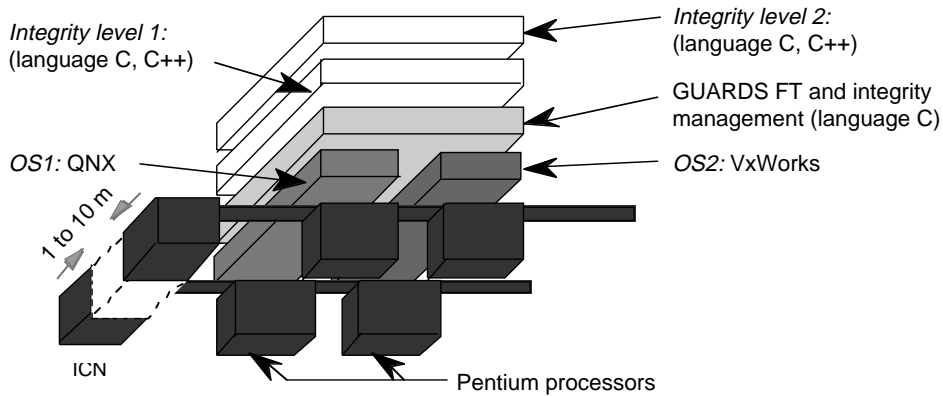
5 • Non-critical monitoring, diagnostic and supervision functions.

6 This is a significant departure from current practice in railway  
7 applications, where these two levels of criticality would normally be  
8 implemented on separate instances. However, there is an appreciable  
9 economic advantage to be gained when it is possible to share the same  
10 hardware between both levels (e.g., for small railway stations).

11 A second railway instance has also been considered for an embedded  
12 train control system and is currently being prototyped. This is a  
13 straightforward duplex fail-safe configuration.

#### 14 **2.9.2 Nuclear Submarine Instance**

15 The targeted nuclear submarine application is a secondary protection  
16 system. The instance considered for this application is a dual-channel  
17 architecture with two Pentium processors in each channel (Figure 8). To  
18 prevent common-mode failures of both channels due to physical damage,  
19 the channels are geographically separated by a distance of several meters.  
20 Like the railway triplex system, this instance hosts two levels of integrity.



1  
2

**Figure 8 — Nuclear submarine duplex instance ( $C=2, M=2, I=2$ )**

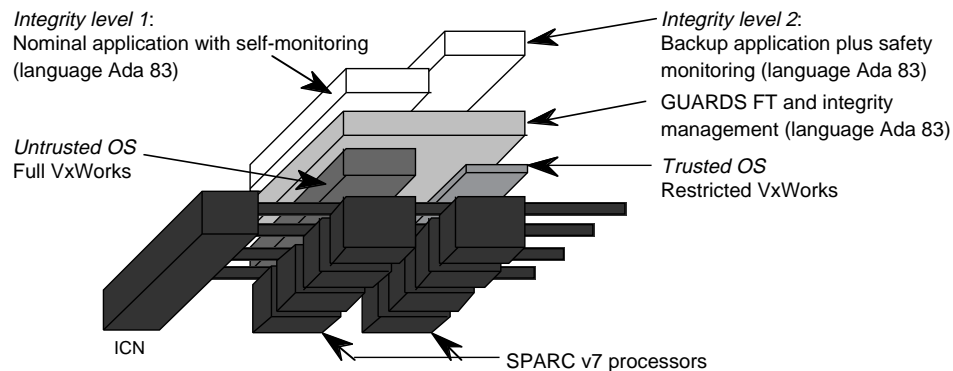
3 An innovative aspect of this instance is the use of two processors in  
4 each channel, with two different POSIX-compliant operating systems:  
5 QNX and VxWorks. Apart from the operating systems, both processors in  
6 each channel run identical application software. The copies of application  
7 components executing on each lane form self-checking pairs to provide  
8 detection of errors due to faults activated independently on each lane. In  
9 particular, this includes physical faults (of the processors) and design faults  
10 of the processors and their operating systems. It is assumed that design  
11 faults of the operating systems are activated independently, based on the  
12 fact that their designs are diversified. Although the processors are identical,  
13 we also assume that faults in their design will be independently activated,  
14 based on their diversification of utilization (due to loose coupling and  
15 diversification of operating systems).

16 As long as both channels are operational, they operate in a two-out-  
17 of-two mode. Results of computations that are declared as error-free by the  
18 intra-channel mechanisms are compared and, in case of disagreement, the  
19 instance is put into a safe state. However, if errors are detected locally, by  
20 intra-channel mechanisms, the channel declares itself to be faulty and the  
21 instance switches to single channel operation. Note that this strategy is

1 different to that of the two-channel configurations of the railway instances  
 2 (duplex instance, or triplex instance degraded to duplex), which switch to a  
 3 safe state whether the error is detected locally or by comparison.

#### 4 **2.9.3 Space Instance**

5 The instance of the architecture for space applications is the most complex  
 6 of those considered. It is a full four-channel instance of the architecture  
 7 capable of tolerating arbitrary faults at the inter-channel level (Figure 9).  
 8 Degradation to three-, two- and one-channel operation is possible. This  
 9 instance also features two levels of integrity.



10

11

**Figure 9 — Space quadruplex instance ( $C=4, M=2, I=2$ )**

12 Like the instance intended for the nuclear submarine application, this  
 13 instance also possesses two lanes, but for a different reason. For the  
 14 nuclear application, the aim was to allow diversified but equivalent  
 15 operating systems to be used so that errors due to design faults could be  
 16 detected. Here, the objective is to have one of the lanes (the *secondary* lane)  
 17 act as a back-up for the other lane (the *primary* lane). Each lane supports a  
 18 different operating system and different application software:





- [Arlat, 1997 #2054] J. Arlat, *Preliminary Definition of the GUARDS Validation Strategy*, LAAS-CNRS, Toulouse, France, Research Report, N°96378, January 1997.
- [Arlat, 1990 #34] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins and D. Powell, "Fault Injection for Dependability Validation — A Methodology and Some Applications", *IEEE Trans. Software Engineering*, 16 (2), pp.166-182, February 1990.
- [Arlat, 1993 #810] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie and D. Powell, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems", *IEEE Trans. Computers*, 42 (8), pp.913-923, August 1993.
- [Audsley, 1993 #2179] N. Audsley, *Flexible Scheduling for Hard Real-Time Systems*, D. Phil. Thesis, Dept. of Computer Science, University of York, UK, 1993.
- [Audsley, 1993 #2029] N. Audsley, A. Burns, M. Richardson, K. Tindell and A. J. Wellings, "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling", *Software Engineering J.*, 8 (5), pp.284-292, 1993.
- [Audsley, 1993 #2182] N. Audsley, K. Tindell and A. Burns, "The End of the Line for Static Cyclic Scheduling?", in *5th Euromicro Workshop on Real-Time Systems*, (Oulu, Finland), pp.36-41, IEEE Computer Society Press, 1993.
- [Barrett, 1995 #2107] P. Barrett, A. Burns and A. J. Wellings, "Models of Replication for Safety Critical Hard Real-Time Systems", in *20th IFAC/IFIP Workshop on Real-Time Programming (WRTP'95)*, (Fort Lauderdale, FL, USA), pp.181-188, Pergamon Press, November 1995.

- [Bates, 1997 #2160] I. Bates and A. Burns, “Schedulability Analysis of Fixed Priority Real-Time Systems with Offsets”, in *9th Euromicro Workshop on Real-Time Systems*, (Toledo, Spain), pp.153-160, IEEE Computer Society Press, 1997.
- [Béounes, 1993 #1407] C. Béounes, M. Aguera, J. Arlat, S. Bachmann, C. Bourdeau, J.-E. Doucet, K. Kanoun, J.-C. Laprie, S. Metge, J. Moreira de Souza, D. Powell and P. Spiesser, “SURF-2: A Program for Dependability Evaluation of Complex Hardware and Software Systems”, in *23rd Int. Conf. on Fault-Tolerant Computing (FTCS-23)*, (Toulouse, France), pp.668-673, IEEE Computer Society Press, June 1993.
- [Bernardeschi, 1998 #2184] C. Bernardeschi, A. Fantechi, S. Gnesi and A. Santone, *Formal Specification and Verification of the Inter-Channel Consistency Network*, PDCC, Pisa, Italy, ESPRIT Project 20716 GUARDS Report, N°I3A4.AO.6009.B, April 1998.
- [Bernardeschi, 1998 #2202] C. Bernardeschi, A. Fantechi, S. Gnesi and A. Santone, *Formal Specification and Verification of the Inter-Channel Fault Treatment Mechanism*, PDCC, Pisa, Italy, ESPRIT Project 20716 GUARDS Report, N°I3A4.AO.6013.A, May 1998.
- [Bernardeschi, 1998 #2185] C. Bernardeschi, A. Fantechi, S. Gnesi and A. Santone, “Formal Validation of Fault Tolerance Mechanisms”, in *Digest of FastAbstracts - 28th Fault-Tolerant Computing Symposium (FTCS-28)*, (Munich, Germany), pp.66-67, 23-25 June 1998.
- [Biba, 1977 #1673] K. J. Biba, *Integrity Considerations for Secure Computer Systems*, The Mitre Corporation, Technical Report, N°MTR-3153, Rev. 1, April 1977.
- [Blough, 1992 #2155] D. M. Blough, G. F. Sullivan and G. M. Mason, “Intermittent Fault Diagnosis in Multiprocessor Systems”, *IEEE Trans. Computers*, 41 (11), pp.1430-1441, November 1992.

- [Bondavalli, 1997 #2094] A. Bondavalli, S. Chiaradonna, F. D. Giandomenico and F. Grandoni, “Discriminating Fault Rate and Persistency to Improve Fault Treatment”, in *27th Int. Symp. on Fault-Tolerant Computing (FTCS-27)*, (Seattle, WA), pp.354-362, IEEE Computer Society Press, June 1997.
- [Bondavalli, 1997 #2106] A. Bondavalli, S. Chiaradonna, F. D. Giandomenico and F. Grandoni, *Inter-Channel State Restoration*, PDCC, Pisa, Technical Note, November 1997.
- [Bondavalli, 1998 #2159] A. Bondavalli, F. D. Giandomenico, F. Grandoni, D. Powell and C. Rabéjac, “State Restoration in a COTS-based N-Modular Architecture”, in *1st Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC'98)*, (Kyoto, Japan), pp.174-183, IEEE Computer Society Press, 20-22 April 1998.
- [Bondavalli, 1997 #2186] A. Bondavalli, I. Mura and M. Nelli, “Analytical Modelling and Evaluation of Phased-mission Systems for Space Applications”, in *2nd Workshop on High Assurance Systems Engineering (HASE-97)*, (Washington, DC, USA), IEEE Computer Society Press, August 1997.
- [Bouali, 1994 #2187] A. Bouali, S. Gnesi and S. Larosa, “The Integration Project for the JACK Environment”, *Bulletin of the EATCS*, October (54), pp.207-223, 1994.
- [Burns, 1995 #2178] A. Burns, N. Hayes and M. F. Richardson, “Generating Feasible Cyclic Schedules”, *Control Engineering Practice*, 3 (2), pp.151-162, 1995.
- [Burns, 1995 #2170] A. Burns and A. Wellings, *HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems*, Real-Time Safety Critical Systems, 3, 313p., Elsevier, 1995.
- [Carreira, 1998 #2188] J. Carreira, H. Madeira and J. G. Silva, “Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers”, *IEEE Trans. Software Engineering*, 24 (2), pp.125-136, February 1998.

- [Dutuit, 1997 #2199] Y. Dutuit, E. Châtelet, J.-P. Signoret and P. Thomas, “Dependability Modelling and Evaluation by Using Stochastic Petri Nets: Application to Two Test Cases”, *Reliability Eng. & System Safety*, 55, pp.117-124, 1997.
- [Fantechi, 1998 #2189] A. Fantechi, S. Gnesi, F. Mazzanti, R. Pugliese and E. Tronci, *A Symbolic Model Checker for ACTL*, PDCC, Pisa, Italy, ESPRIT Project 20716 GUARDS Report, N°I3A5.AO.6011.A, April 1998.
- [Gong, 1998 #1992] L. Gong, P. Lincoln and J. Rushby, “Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults”, in *Dependable Computing for Critical Applications 6*, (R. K. Iyer, M. Morganti, W. K. Fuchs and V. Gligor, Eds.), Dependable Computing and Fault-Tolerant Systems, 10, pp.139-157, IEEE Computer Society Press, 1998 (Proc. IFIP 10.4 Work. Conf. held in Urbana-Champaign, IL, USA, September 1995).
- [Gray, 1986 #1234] J. Gray, “Why do Computers Stop and What can be done about it?”, in *5th Symp. on Reliability in Distributed Software and Database Systems*, (Los Angeles, CA, USA), pp.3-12, IEEE Computer Society Press, January 1986.
- [Harper, 1990 #1978] R. E. Harper and J. H. Lala, “Fault-Tolerant Parallel Processor”, *Journ. of Guidance, Control and Dynamics*, 14 (3), pp.554-563, May-June 1990.
- [HOOD, 1992 #2172] *HOOD Reference Manual*, Release 3.1.1, HOOD Technical Group, 1992.
- [Hsueh, 1997 #2100] M.-C. Hsueh, T. K. Tsai and R. K. Iyer, “Fault Injection Techniques and Tools”, *IEEE Computer*, 40 (4), pp.75-82, April 1997.
- [IEC 1508, #2122] *Functional Safety: Safety-Related Systems*, Draft International Standard IEC 1508, International Electrotechnical Commission, IEC Document N°65A/179/CDV (Geneva, June 1995).

- [Intecs, 1996 #2171] *HRT-HoodNICE: a Hard Real-Time Software Design Support Tool*, Intecs Sistemi, Pisa, Italy, ESTEC Contract 11234/NL/FM(SC), Final Report, 1996.
- [Jenn, 1998 #2191] E. Jenn, *Modelling for Evaluation*, Technicatome, Aix en Provence, France, ESPRIT Project 20716 GUARDS Report, N°I3A3.TN.0056.A, January 1998.
- [Kanoun, 1996 #2080] K. Kanoun, M. Borrel, T. Morteveille and A. Peytavin, "Modelling the Dependability of CAUTRA, a Subset of the French Air Traffic Control System", in *26th Fault-Tolerant Computing Symp. (FTCS-26)*, (Sendai, Japan), pp.106-115, IEEE Computer Society Press, June 1996.
- [Kieckhafer, 1988 #766] R. M. Kieckhafer, C. J. Walter, A. M. Finn and P. M. Thambidurai, "The MAFT Architecture for Distributed Fault Tolerance", *IEEE Trans. Computers*, 37 (4), pp.398-405, April 1988.
- [Krishnamurthy, 1998 #2192] N. Krishnamurthy, V. Jhaveri and J. A. Abraham, "A Design Methodology for Software Fault Injection in Embedded Systems", in *Proc. IFIP Int. Workshop on Dependable Computing and Its Applications (DCIA-98)*, (Y. Chen, Ed.), (Johannesburg, South Africa), pp.237-248, Springer, January 1998.
- [Lala, 1994 #1884] J. H. Lala and R. E. Harper, "Architectural Principles for Safety-Critical Real-Time Applications", *Proc. IEEE*, 82 (1), pp.25-40, January 1994.
- [Lamport, 1985 #239] L. Lamport and P. M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults", *Journ. of the ACM*, 32 (1), pp.52-78, January 1985.
- [Laprie, 1995 #2140] J.-C. Laprie, J. Arlat, J.-P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.-C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac and P. Thévenod, *Dependability Guidebook*, 324p., Cépaduès-Editions, Toulouse, 1995.

- [Lee, 1990 #1509] S. Lee and K. G. Shin, “Optimal Multiple Syndrome Probabilistic Diagnosis”, in *20th Int. Symp. on Fault-Tolerant Computing Systems (FTCS-20)*, (Newcastle upon Tyne, UK), pp.324-31, IEEE Computer Society Press, 1990.
- [Leung, 1982 #2164] J. Leung and J. Whitehead, “On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks”, *Performance Evaluation*, 2 (4), pp.237-250, 1982.
- [Lincoln, 1993 #1444] P. Lincoln and J. Rushby, “A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model”, in *23rd Int. Conf. on Fault-Tolerant Computing (FTCS-23)*, (Toulouse, France), pp.402-411, IEEE Computer Society Press, 1993.
- [Lundelius-Welch, 1988 #1020] J. Lundelius-Welch and N. Lynch, “A New Fault-Tolerant Algorithm for Clock Synchronization”, *Information & Computation*, 77 (1), pp.1-16, 1988.
- [Melliari-Smith, 1982 #270] P. M. Melliari-Smith and R. L. Schwartz, “Formal Specification and Mechanical Verification of SIFT: A Fault-Tolerance Flight Control System”, *IEEE Trans. Computers*, C-31 (7), pp.616-630, July 1982.
- [Owre, 1996 #2198] S. Owre, S. Rajan, J. M. Rushby, N. Shankar and M. K. Srivas, “PVS: Combining Specification, Proof Checking, and Model Checking”, in *Computer-Aided Verification (Proc. CAV'96, New Brunswick, NJ, USA, July/August 1996)*, (R. Alur and T. A. Henzinger, Eds.), LNCS, 1102, pp.411-414, Springer-Verlag, New-York, USA, 1996.
- [Paganone, 1997 #2169] A. Paganone and P. Coppola, *Specification and Preliminary Design of the Architectural Development Environment*, Intecs Sistemi, Pisa, Italy, ESPRIT Project 20716 GUARDS Report, N°D2A1.A0.3002.C, April 1997.
- [Pease, 1980 #1937] M. Pease, R. Shostak and L. Lamport, “Reaching Agreement in the Presence of Faults”, *Journ. of the ACM*, 27 (2), pp.228-234, April 1980.

- [Poledna, 1998 #2027] S. Poledna, “Deterministic Operation in of Dissimilar Replicated Tasks Sets in Fault-Tolerant Distributed Real-Time Systems”, in *Dependable Computing for Critical Applications 6*, (M. Dal Cin, C. Meadows and W. H. Sanders, Eds.), pp.103-119, IEEE Computer Society Press, 1998 (Proc. IFIP 10.4 Work. Conf. held in Grainau, Germany, March 1997).
- [Powell, 1994 #1730] D. Powell, “Distributed Fault-Tolerance — Lessons from Delta-4”, *IEEE Micro*, 14 (1), pp.36-47, February 1994.
- [Powell, 1997 #2012] D. Powell, *Preliminary Definition of the GUARDS Architecture*, LAAS-CNRS, Toulouse, France, Research Report, N°96277, January 1997.
- [Powell, 1998 #2162] D. Powell, J. Arlat and K. Kanoun, *Generic Architecture Instantiation Guidelines*, LAAS-CNRS, Toulouse, France, Research Report, N°98136, May 1998.
- [Powell, 1998 #2181] D. Powell, C. Rabéjac and A. Bondavalli, *Alpha-count Mechanism and Inter-Channel Diagnosis*, ESPRIT Project 20716 GUARDS Report, N°I1SA1.TN.5009.E, 1998.
- [Rabéjac, 1997 #2153] C. Rabéjac, *Inter-Channel Fault Treatment Mechanism*, Matra Marconi Space, France, Guards Report, N°D1A3 AO 2014 B, March 1997.
- [Ramanathan, 1990 #1017] P. Ramanathan, K. G. Shin and R. W. Butler, “Fault-Tolerant Clock Synchronization in Distributed Systems”, *Computer*, pp.33-42, October 1990.
- [Sanders, 1993 #1459] W. H. Sanders and W. D. Obal II, “Dependability Evaluation Using UltraSAN”, in *23rd Int. Conf. on Fault-Tolerant Computing (FTCS-23)*, (Toulouse, France), pp.674-679, IEEE Computer Society Press, 1993.

- [Schwier, 1997 #2195] D. Schwier and F. von Henke, “Mechanical Verification of Clock Synchronization Algorithms”, in *Design for Validation*, ESPRIT Long Term Research Project 20072: DeVa - 2nd Year Report, pp.287-303, LAAS-CNRS, Toulouse, France, 1997.
- [Semini, 1998 #2196] L. Semini, *Formal Specification and Verification for an Integrity Policy Supporting Multiple Levels of Criticality*, PDCC, Pisa, Italy, ESPRIT Project 20716 GUARDS Report, N°I3A5.AO.6012.A, April 1998.
- [Simpson, 1990 #2177] H. Simpson, “Four-Slot Fully Asynchronous Communication Mechanism”, *IEE Proc*, 137, Pt. E (1), pp.17-30, January 1990.
- [Srikanth, 1987 #353] T. K. Srikanth and S. Toueg, “Optimal Clock Synchronization”, *Journ. of the ACM*, 34 (3), pp.626-645, July 1987.
- [Thambidurai, 1988 #1993] P. Thambidurai and Y.-K. Park, “Interactive Consistency with Multiple Failure Modes”, in *7th Symp. on Reliable Distributed Systems (SRDS-7)*, (Columbus, OH, USA), pp.93-100, IEEE Computer Society Press, 1988.
- [Tindell, 1993 #2180] K. Tindell, *Fixed Priority Scheduling of Hard Real-Time Systems*, D. Phil. Thesis, Dept. of Computer Science, University of York, UK, 1993.
- [Total, 1998 #2165] E. Total, L. Beus-Dukic, J.-P. Blanquart, Y. Deswarte, D. Powell and A. Wellings, “Integrity Management in GUARDS”, in *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, (Lake District, England), Chapman and Hall, 15-18 September 1998.
- [Total, 1998 #2151] E. Total, J.-P. Blanquart, Y. Deswarte and D. Powell, “Supporting Multiple Levels of Criticality”, in *28th Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, (Munich, Germany), pp.70-79, IEEE Computer Society Press, 23-25 June 1998.

[Wellings, 1997 #2173] A. Wellings and L. Beus-Dukic, *Guidelines for Mapping HRT-HOOD to POSIX/C*, University of York, UK, ESPRIT Project 20716 GUARDS Report, N°I2A1-2.A0.7041.B, December 1997.

[Wellings, 1998 #2163] A. Wellings, L. Beus-Dukic and D. Powell, *Real-Time Scheduling in a Generic Fault-Tolerant Architecture*, LAAS-CNRS, Research Report, N°98162, May 1998.