

1 Chapter 1

2 **Requirements and Motivations**

3 The development and validation of fault tolerant computers for
4 critical real time applications are currently both costly and time-consuming.
5 Often, the underlying technology is out of date by the time the computers
6 are ready for deployment. Facing these obsolescence issues, the
7 requirements and objectives of the GUARDS project are determined by the
8 fundamental motivation of introducing Commercial Off-the-Shelf (COTS)
9 components in Safety Critical and Safety Related systems, allowing them
10 to take benefit of the fast progress of technology.

11 More than a just a new kind of architecture, a generic approach is
12 sought, which would encompass architecture, technology and design
13 methodologies. The GUARDS approach, which is aimed at covering all
14 these aspects, is a global system-engineering approach. After setting out
15 this fundamental motivation, this chapter intends to describe both the end
16 users' needs and the subsequent requirements.

17 **2.1 Fundamental motivation**

18 In a world in which business becomes more and more difficult,
19 minimising the overall cost of a project improves company
20 competitiveness. This is particularly true for the design of dependable
21 computing systems which have a lifetime of several decades. Analysing

1 this overall cost implies the analysis of the life-cycle of such systems, and
2 the kind of maintenance operations needed to maintain the service.

3 The initial *development* stage concerns the implementation of the
4 functional specification of an application in a computer, fulfilling the
5 specified dependability requirements.

6 During the product life, it may be necessary to add new functionality
7 which may require more powerful CPUs and other new components. This
8 may lead to an *upgrade* of the computer with up-to-date COTS hardware
9 and software. Similarly, an upgrade of the computer may be required to
10 address the issue of the rapid obsolescence of its components.

11 Hence, the targeted costs are related to the whole life-cycle of a
12 product: development cost at the beginning of its life and the cost of each
13 modification, including re-validation.

14 For *embedded real time systems* with a *high level of dependability* an
15 important effort is placed on the validation tasks. Furthermore, the required
16 high levels of dependability and performance of these systems cannot be
17 provided by standard commercial products.

18 Besides that, frequent updates may be required by the very short life-
19 cycle of these commercial products; if the non functional requirements
20 (dependability, real-time performance) are provided in a specific way, at the
21 application level, this could lead to very high modification costs, because of
22 the re-validation operations required.

23 The reuse of validated parts is also suitable from one application to
24 another, to lower the development costs

25 Hence, to take benefit of the use of COTS components, and actually
26 reduce the life cycle cost, it is necessary to define an approach helping to:

- 27 • Fulfil various levels of dependability requirements, depending on the
28 targeted application, or the phase of mission of the system.
- 29 • Support the hard real time constraints of systems for which the respect
30 of deadlines is a condition of the mission success.

- 1 • Reuse validation results for parts of systems that are reused from one
2 application to another or in case of an upgrade.

3 **2.2 Application domains**

4 The architecture is aimed at two major classes of application:

- 5 • Critical instrumentation.
6 • Safety-critical or safety-related process control.

7 The targeted application domains are primarily those of the
8 consortium's end user partners, which are diverse and rich enough to
9 validate the approach.

- 10 • Railway applications.
11 • Nuclear applications.
12 • Space applications.

13 Potentially, however, GUARDS can address the same application
14 classes for other application domains:

- 15 • Avionic applications.
16 • Defence applications.
17 • Chemical, petroleum and processing industry applications.

18 We describe below some typical railway, nuclear and space
19 applications, along with their main characteristics.

20 **2.2.1 Railway applications**

21 In the railway domain, we can define five distinct areas of application of
22 GUARDS to different signalling system functions and products.

- 23 • Communication technologies, especially track to train.

- 1 • Information processing technologies, especially in safety-critical
2 applications.
- 3 • Signalling system component technologies.
- 4 • Signal and train system architecture.
- 5 • Process technologies used in developing signalling and train control
6 products.

7 Standards in this domain dictate extremely low catastrophic failure
8 rates for individual subsystems (e.g., less than 10⁻¹¹/hour with respect to
9 physical faults). In railway applications, it is common to physically
10 segregate subsystems responsible for vital (safety-critical) functions from
11 non-vital functions. It could be useful to investigate the possibility of a
12 single instance supporting both high-integrity vital functions and low-
13 integrity non-vital functions. This would be effective in reducing costs.

14 **2.2.2 Nuclear applications**

15 In the nuclear domain, GUARDS targets mainly the Instrumentation
16 and Control (I&C) of nuclear reactors for power generation of a nuclear
17 propulsion system for submarines. The I&C subsystem is critical for the
18 operational mission, and is classified B according to IEC1226 (availability
19 of power generation). The relevant part of the I&C subsystem is related to
20 process regulation, command and control of all the actuators that are critical
21 for the availability of power generation, and all the man/machine interfaces
22 in the control room.

23 Other targeted projects include:

- 24 • Experimental reactors.
- 25 • Nuclear I&C revamping (i.e., a major upgrade in the product life)

26 These kinds of application lead to very long mission duration
27 between possible maintenance operations.

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

7 **2.2.3 Space applications**

8 In the space domain, the needs for a systematic provision of dependability
9 for embedded computer system are encountered in highly critical real-time
10 functions embedded in space systems devoted to long life time missions or
11 manned transportation systems.

12 Examples of such applications are:

- 13 • Autonomous spacecraft and long life time scientific probe
14 (ROSETTA).
- 15 • Space system tolerant to very harsh environments (military satellites or
16 robots).
- 17 • Automatic systems performing rendezvous with manned systems
18 (ATV i.e. Automated Transfer Vehicle).
- 19 • Manned systems space vehicles and platforms (CTV, i.e., Crew
20 Transfer Vehicle and CRV, i.e., Crew Rescue Vehicle).

21 A particularly challenging application in the space domain is that of
22 an autonomous spacecraft carrying out missions containing phases that are
23 so critical that tolerance of several faults may be required (e.g., target fly-by
24 or docking). During non-critical phases, the redundant elements may be
25 powered down to save energy. Moreover, like the railway application, it is
26 necessary for an instance to be able to support software of different
27 integrity levels: high-integrity critical software that is essential for long-
28 term mission reliability and potentially unreliable payload software.

1

2 **2.3 Requirements**

3 The main requirements follow from the two characteristics of the targeted
4 applications: real-time and dependability, and from the fundamental
5 motivation: reducing the overall cost of these applications. The idea is to
6 define a set of mechanisms (hardware and software) with well-defined
7 interfaces, which constitute a generic fault-tolerant architecture with focused
8 validation obligations. The functional components and the fault tolerance
9 mechanisms should be as separated as possible in the implementation and
10 it should be possible to validate them separately.

11 Real-time constraints and dependability requirements not only have
12 an impact on the architecture (and on the technology), but lead also to
13 requirements related to the development process.

14 We first state the requirements from the dependability viewpoint and
15 then those related to the architecture, to the development process and to
16 technology.

17 **2.3.1 Dependability**

18 The architecture addresses safety-critical and safety-related systems.
19 The deployment of such systems requires an agreement of the safety
20 authorities of the targeted domain. The main dependability requirements are
21 thus related to:

- 22 • Certification.
- 23 • Criticality levels of the supported functions.
- 24 • Fault models taken into account.
- 25 • Validation.

1 **2.3.1.1 Certification**

2 The architecture proposed by the GUARDS approach must allow the
3 compliance of the resulting computer with domain-specific certification
4 standards, despite the use of COTS components, in one of two ways:

- 5 • By the use of certified COTS components.
- 6 • By the use of mechanisms and an architecture that allow the certification
7 of the computer despite the use of non-certified COTS components.

8 The approach adopted for achieving dependability must not rely on
9 diversity of application software. However, the GUARDS approach will
10 not prevent the use of such diversity.

11 **2.3.1.2 Criticality levels**

12 It could be useful (i.e., when different applications share the same
13 hardware) to allow the coexistence of functions in a single computer with
14 different levels of criticality. The coexistence of functions of different
15 criticality requires two levels of segregation:

- 16 • At the hardware level (e.g., between I/O boards sharing the same back
17 plane),
- 18 • At the software level (e.g., between functions sharing the same CPU,
19 and possibly communicating with each other).

20 **2.3.1.3 Fault models**

21 The architecture should be tolerant of the following classes of faults:

- 22 • Independent physical faults of components (as defined by the usual
23 databases, e.g., CNET RDF 93), including those leading to Byzantine
24 behaviour.

- 1 • Stochastically dependent faults (common mode, common cause and
2 cascade faults), especially:
- 3 – Common-cause faults due to the aggressions from the
4 environment.
- 5 – Correlated design faults.
- 6 Obviously, investigation concerning dependent faults will be limited
7 to the current state-of-the-art. Man-machine faults are not taken into
8 account.
- 9 The set of mechanisms proposed by the approach should include a
10 mechanism to tolerate transient faults.

11 **2.3.1.4 Validation**

12 Validation encompasses functional validation and dependability properties
13 demonstration.

14 Assuming that the functional validation is application dependent, the
15 demonstration of the dependability properties can rely on two
16 complementary approaches:

- 17 • Analytical validation, aimed at verifying and evaluating the basic
18 foundations of the architecture.
- 19 • Experimental validation, by fault injection, by exploitation of field
20 experience, etc, to assess the dependability of particular instances of the
21 architecture.

22 In the framework of the analytical approach, the validation of the
23 specifications of dependability requirements may include both model-based
24 evaluation and formal methods. This approach is aimed at making an
25 adequate and justified valuation of the key parameters for a particular
26 application. In that way, it contributes to ensure the compliance with the
27 dependability requirements.

1 **2.3.2 Architecture**

2 The GUARDS approach aims to provide an upgradable architecture able to
3 be reconfigured in case of failure, with the necessary and efficient
4 mechanisms to detect fault occurrences and if possible to limit their effects.
5 This chapter will state the requirements related to real time, to modularity
6 and reconfiguration, and to input and output management.

7 **2.3.2.1 Real time**

8 The respect of the dependability requirements with functional constraints
9 like deadlines implies *a priori* confidence in the ability of the system to
10 respect these constraints during normal operation (i.e., without faults). To
11 this end, we need a method aimed at specifying and verifying the real time
12 behaviour of an application, i.e., the respect of a short and predictable
13 response time.

14 The tolerance of timing faults will be achieved through the
15 monitoring and control of the behaviour of the application in order to detect
16 fault occurrences and, if possible, to limit their effects (for example, by
17 killing the faulty process). To this end, a set of GUARDS mechanisms will
18 be provided for application self-checking operations, such as:

- 19 • Deadline self-check: monitoring the total delay of a task between its
20 activation and its termination.
- 21 • Resource monitoring: verifying the actual processing time used by a
22 task.

23 **2.3.2.2 Modularity**

24 The use of the same architecture in various application domains, favours
25 the use of a configurable hardware responding to various safety, availability
26 and performance objectives.

- 1 The choice of a redundancy strategy is driven by the dependability
2 objectives. For example, the safety may be performed by a 2 out of 2
3 architecture, while 1 out of 2 is used for availability and 2 out of 3 for both
4 availability and safety.
- 5 Tuning the processing power, depending on the application is also suitable.
- 6 Hence, a modular approach is required that allows the definition of fault-
7 tolerant instances with various redundancy strategies (1/2, 2/2, 2/3, or 2/4),
8 and various processing capabilities with as minimal adaptations as possible
9 of the application software.
- 10 A preliminary analysis leads to the following requirements:
- 11 • A means to define *redundant entities* in the fault tolerant computer
12 architecture, and a set of mechanisms allowing error detection, partial
13 degradation (isolation of a redundant entity), and “self-repair” of the
14 fault-tolerant computer by reintegration of a transiently-faulty entity.
 - 15 • A set of mechanisms to *consolidate* the different redundant entities at
16 the processing level (results comparison between processing units) and
17 at the input/output level (input and output consolidation).
 - 18 • A means to *monitor real-time behaviour* (scheduling and time
19 consumption for each task).

20 **2.3.2.3 Reconfiguration**

21 Performing reconfiguration of a fault-tolerant computer by passivation of
22 its redundant entities allows it to continue to deliver the application services,
23 in case of a fault occurrence. The new configuration of the instance
24 resulting from this reconfiguration may be associated to specific functional
25 or dependability requirements; this constitutes a "degraded mode". A
26 degraded mode may consist in a reduced set of available services, degraded
27 dependability, or degraded performance.

28 It is very important, from the mission point of view, that the
29 reconfiguration or the replacement of any part of the fault-tolerant computer

1 does not induce interruptions in the delivery of the critical services. For
2 example, isolation of a redundant entity implies disconnection of the related
3 outputs until its reintegration may be performed (i.e., after a successful
4 testing operation).

5 For systems with a very long mission time, the reconfiguration of
6 the fault-tolerant computer may be performed manually by an operator (on
7 the basis of the diagnosis information provided by the system).

8 **2.3.2.4 Inputs/Outputs**

9 A typical process control application acquires information on the state and
10 the evolution of the physical process through sensors and remote
11 computing devices. Data may be provided by redundant sources.

12 The fault-tolerant computer has to check the validity (consistency,
13 timeliness, etc.) of data received from the sensors and other logical agents.
14 This information has to be consolidated (by comparison of information
15 acquired at the same time) and any possible inconsistency identified, if
16 possible by application-independent software.

17 **2.3.3 Development process**

18 The GUARDS system software will have to be interfaced to application
19 modules written in different languages. The application development
20 process used by different end-users may be based on different tools:

- 21 • Different standard application design tools.
- 22 • Different extension tools for extracting the information needed for
23 monitoring of the real time tasks.
- 24 • Different validation tools.

25 Different methods are used at the application level, depending on the
26 field of application ; indeed, the methods are tightly adapted to the
27 applications (for example : the specification methods are different in the

1 spatial , railway and nuclear domains). A common and unique
2 development environment is not required. But, it is suitable that these
3 different tools be able to communicate with each other and together
4 constitute a consistent engineering environment.

5 In the framework of the validation tools, the modelling tasks require
6 an environment able to propose a set of pre-defined models and a means to
7 specify new models adapted to the application. Experimental validation by
8 fault injection requires also specific tools supporting the whole fault
9 injection campaign : injection points specifications, injection scenarios,
10 results logging and storing....

11 **2.3.4 Technology**

12 The best flexibility of the proposed architecture will be obtained with a set
13 of mechanisms that are as independent as possible from the chosen COTS
14 components. For example, the GUARDS system software should aim to
15 use a common set of services proposed by the chosen real-time operating
16 system. The cost of the adaptations to another operating system will be as
17 limited as possible.

18 The interface between the mechanisms and the COTS components
19 will be a subset of well-defined standards. Sticking to standards does not
20 guarantee the life-time of a product, but this approach a priori guarantees a
21 certain level of reproducibility and the capability for the compliance of new
22 components to be checked. For example, if POSIX becomes obsolete
23 sooner than expected, an emulation of the services should still be possible
24 on the basis of a new standard.

25 The fault-tolerant architecture will be designed with the same rules of
26 segregation at the hardware level as at the software level. For example,
27 segregation of the power source of each redundant entity contributes to the
28 fault containment strategy.

29

1 **2.3.5 The GUARDS approach**

2 To summarise, the generic approach sought should propose :

- 3 • A method to ensure by construction the respect of timeliness and
4 predictability of real-time behaviour of the application and to
5 demonstrate a priori the compliance with real-time requirements.
- 6 • A scalable, configurable architecture both with respect to dependability
7 (safety, availability) and computing resources.
- 8 • Dependability validation means: fault-injection tools to verify that the
9 GUARDS mechanisms operate properly and a dependability modelling
10 environment to ensure the compliance with the dependability
11 requirements.

12 .

13