DEVELOPMENTS IN A NUCLEAR POWER STATION PROGRAMMED PROTECTION SYSTEM

Eric PILAUD
Merlin Gerin
Département SES
38050 GRENOBLE
FRANCE

ABSTRACT

This paper aims at presenting, by examining the development of nuclear power stations programmed protection systems, the problems and challenges set by the safety requirements of these systems.

Between 1980 and 1984 Merlin Gerin, in collaboration with Framatome and Electricité de France, developed the first nuclear power stations programmed protection system (SPIN) for the French 1300 MW power stations. It was necessary to deal with the problem of software safety. A safety oriented software was adopted and applied to these systems. These systems have been in service in more than 20 sites during 6 years, enabling a balance to be drawn in our favour.

Since 1986, a new generation of the SPIN system for 1450 MW plants, which corresponds to a significant expansion of the software, has been developed.

This requires certain improvements in order to overcome increasing complexity and bring some questions for the future...

1. HARDWARE ARCHITECTURE

The hardware SPIN architecture is very classical, using massive (4MR voting scheme) and selective (self tests, error correcting codes...) redundancies. The architecture of the last generation of SPIN (for the new 1450 MW plants called N4 series) is briefly presented in the first part of this paper.

The utilization of computer systems in such critical sectors raises the crucial problem of software safety. The design of critical software in the nuclear field is subject to IEC 880 Standard [1]. The development methodology used in Merlin Gerin, which relies on this standard, is presented in section 2.

The results obtained with this methodology in the development of French 1300 MW power stations have already been published [2]. They are briefly resumed in section 3.

But the new generation of the SPIN corresponds to a technological development based essentially on the use of local networks and on a significant expansion of the software. The necessity of overcoming the complexity leads us to adapt the methodology and presents questions for the future. These points are discussed in section 4 and in the conclusion.

0. INTRODUCTION

MERLIN GERIN has been producing, in collaboration with FRAMATOME and ELECTRICITÉ DE FRANCE two generations of programmed core protection systems, called SPIN, for the French nuclear power stations.

Using thermodynamic sensors, neutron detectors and rod position sensors, the SPIN (Integrated Digital Protection System) controls emergency shutdown of the reactor to avoid accidental damage and some safeguard functions (such as containment spraying, emergency feed water, ...).

Safety requirements of such a system (not missing a shutdown) are very high, but are matched, however, by availability requirements, since unscheduled shutdowns are very expensive and could disturb the running of the core.

The SPIN is composed of subsystems:

- the UATP (Acquisition and Processing Unit for Protection) with redundancy 4
- the ULS (Logic Safeguard Unit) with redundancy 2 for controlling safeguards actions and emergency shutdown.

Redundancy 4 of the UATP enables a periodic test to be carried out without affecting the availability of the plant; so the system can function in the following states:

- tripping in 2/4 without test; in 2/3 during test
- tripping in 2/3 without test and one defective channel (sensor or UATP); in 1/2 during test and one defective channel.
The periodic testing system is composed of a central test unit (UTC) and local test units for each UATP (UTLP) and for each ULS (UTLS).

The SPIN system is connected to two Transfer and Diagnostic Units (UTD).

The UATP-ULS and UATP-UTD links are made by local, redundant, optic fibre networks, given over entirely to protection (i.e. 4x2 networks) ; these networks are especially designed for safety requirements (self tests and error correcting codes). This type of network is named NERVIA.

The ULS-UTD links are made by Nervia type networks, dedicated to signalling (i.e 2 networks).

The 2/4 voting logic, sensor inhibitions or sensor safe positioning and inhibition of UATP are accomplished by the ULS.

The first level of architecture is presented in figure 1.

The UATP are themselves partially redundant:

- each UATP acquires analog, boolean and pulse signals in two redundant acquisition units UA (so that, for each input, one active and one spare signal is available). Processing (filtering, threshold comparisons, protection algorithms...) are accomplished by five diversified functional units (UF) in each UATP.

In the same way, each ULS is composed of 4 processing units in redundance.

All these units are based on a 68000 CPU with NERVIA stations performing continuously several self tests (watchdogs, invalid codes detections, memory checks, ...).

Finally, let us point that all the hardware is designed in conformance with IEC standards and is especially robust regarding electromagnetic compatibility requirements and seismic requirements.

2. SAFETY ORIENTED SOFTWARE METHODOLOGY

The development methodology used is based on three major points:
- the strict adherence to a specific life cycle model, with clearly defined steps and the production of qualified documents at each step,
- the setting up of a verification/validation team who is independent from the design team, and who is responsible for the verification and qualification of the documents produced at each step before authorization is given for the progression to the next,
- the adherence to a set of design and programming rules and recommendations whose purpose is to eliminate hazardous, over complicated or insufficiently legible constructions.

2.1. Life Cycle

The software is design and qualified, step by step, according to the conventional "V life cycle" (figure 2).

During the specification phase, the software descriptive document and the software assessment and test document (black box tests description) are produced and several other documents are initiated: Software Assurance Quality Plan, User Manual, Software Documents List,...

During the design phases, design documents and corresponding test description documents are written.

During the tests phases (unit tests, integration, validation) all the results of tests are collected.
2.2. The verification/validation team

This team can exercise a veto on any of the design phases carried out by the design team. Its task is threefold:

- to assess the design documents:
  
  this task is to locate the following errors: inconsistency with the source documents, inconsistencies within the document, premature choices of design, ambiguous descriptions, over sophisticated or dangerous constructions, incomplete documents, inconsistency with the design rules, documents written in such a way as they might create problems for maintenance. Note that this task includes re-reading of the code.

- to define the unit tests and validation tests:
  
  entrusting these tasks to people who are not involved in the design of the software is efficient and rewarding for two reasons: first, attempting to define tests using a design document, written by somebody else, leads one to be very critical concerning its contents and second, it enables a true "black box" approach.

- to run these tests

2.3 Rules for design and programming

This simply means the establishment, application and awareness of a set of rules intended to limit the complexity and the risks of side effects in the programs. These rules are described in the IEC880 standards and in some internal documents (for example: rules for programming in C language, rules for establishment of preliminary and detailed design documents,...)

The following list gives only a few examples:

- the restricted use of interrupts and recursiveness
- strict rules concerning the use of pointers and global variables
- recommendations concerning the size of modules, the depth of nesting, the number of branches,...
- the utilization of defensive programming

3. RECAP OF THE RECORD OF SPIN P4 (1300 MW PLANTS)

This system was the first using software in a protection system. It was developed between 1980 and 1984.

The software was developed in assembler 6800 code. The first version consisted of 40000 lines of assembler code and it took 13400 hours of development and validation time.

During 6 years of use at more than 20 sites, 9 successive versions was developed leading to a small extension of the size of the software. 5 minor software failures was detected. Not a single failure affecting the protection has been observed.

During development, errors was detected very early, as shown in figure 3.

![Figure 3](image)

4. EVOLUTION BETWEEN THE TWO GENERATION OF SYSTEMS

The new generation (SPIN N4 for 1450 MW plants) has been developed between 1986 and 1990 (version 0 of the software). It is characterized by three notable technological evolutions:

- the use of the 68000 microprocessor
- the use of the C language
- the intensive use of communications by local networks.

One consequence of these evolutions has been that the complexity of the software has increased significantly: the possibility of more extensive communications and better performances have given rise to the definition of supplementary functions and new characteristics:

- more communications with the other core control command systems
- communications with the diagnostic units
- the software must be "expandable" and be able to accept some supplementary inputs and outputs without having to perform any major software modifications.
However, the increase in complexity of the software has resulted in a large growth in development costs and the size of development and validation teams:

- about 90000 hours have been spent for the development and validation of the SPIN software,
- due to the need for extensive communications with other systems, the SPIN software team could not be managed alone and was incorporated into a larger team responsible for producing all the of the core control and command system,
- at its largest, the SPIN software team was represented by 21 members (design and verification) integrated into a software team of 46 members.

Faced with this development, the efficiency of the simple methodology applied to the 1300 MW SPIN could be brought into question. This methodology is based on the capacity of human being to:

- discipline himself to observe the formalized design procedures
- discipline himself to produce documents during design
- discipline himself to check the design constantly, including after each modification.

Facing with this analysis at the beginning of the project, our approach was the following: if it was possible to trust a small group of people to respect these constraints and that it was actually possible to check their work efficiently for SPIN 1300 MW, it was doubtful that the same applies in the new context.

For this reason, it was decided to develop and use a software tool, called SAGA, intended to improve the strict application of the safety oriented software methodology. This tool, presented in more details in [3], supports the specification, design and code generation phases; it is based on:

- the use of a specification language with precise formal semantics: this language is derived from the LUSTRE language, developed at the University of Grenoble [4]. This is a synchronized data flow language which can be seen as a sub-assembly of linear temporal logic.
- an interactive graphic support of this language
- a top-down design by successive refinements

From the safety point of view, the main characteristics are:

- using a language based on a strong formal basis avoid ambiguities
- the tool comprises restrictions which cannot be overridden by the designer and enforce him to follow the methodology (for example, it is impossible to proceed to the next step unless everything has been documented and declared)

In the same way, some other tools are used in order to provide an automatic support for some tedious tasks (the first which are “forgotten” or neglected! and in order to force the team members to respect the methodology (complexity analysis tool, automatic testing tools, ...)

At the moment this paper is written, it is still difficult to present a complete outcome of the 1450 MW SPIN development. Some features can already be mentioned:

- 170000 lines of C code have been produced for the SPIN system
- 30000 lines of assembler 68000 has been produced for the low-level functions (functions very close to the hardware)
- about 53000 hours have been spent by the design team and 37000 hours by the verification and validation team
- the use of the SAGA tool and the effort spent on the verification and validation of the simple functions allow the integration phase and the validation phase to be very short, regarding the size of the software

5. CONCLUSION

The approach presented here seems efficient. The use of tools seems necessary in order to help (and force) the team to apply the procedures and rules prescribed by the methodology.

Some questions for the future seem still unanswered:

- how to place this “Design Team/Verification Team approach in relation with the “N-Versions design” approach:

  on one hand, due to the increasing complexity, the cost of verification is rising, causing the first approach to lose some of its economic benefits.
on the other hand, in our field, the efficiency of the second approach seems to clash with the difficult problem of common mode failures (due for example to common specification document);

- the problem of software reliability quantification remains unanswered in our case:

we have tried to apply some models, based on the observation of failures occurring during tests; because of the low number of errors encountered during execution (this is due to the efficiency of the methodology) the extrapolations from the error readings remain perplexing

- a restrictive use of formal proofs seems promising: it must be possible in the future, using formal specification, to derive, throughout the design process, simple properties that the program must check to respect the safety requirements. In this manner, it is hoped that the safety attribute can be proved (or partially proved) without having to formally prove the whole of the program, which is today practically impossible

REFERENCES


