Extended Abstract

AUBRY Jules

Toulouse INP
jules.aubry@etu.toulouse-inp.fr

Abstract. Safety Critical Systems are omnipresent in various sectors from transportation to nuclear energy where any fonctionnal failure is unacceptable. Face of the multiplicity of existing way to fiabilize this systems, our objetive was to realize a complete state of the art and categorize these methods.
To do this, we analyzed many scientific articles which presented technologies applied to specific domains. Then, we evaluated their efficiency in other envirronments. This approach has allowed us to propose some comparative tables of programming languages adapted to each field and for specific needs of critical systems while highlighting a potentially generalizable method.
This paper offers to novice an overview of the sector, that allows us to orient themselves and select the right solution for their requirements.

Keywords: Safety Critical Systems, Survey, Formal Methods, Modelling and Verification

1 Introduction

Safety Critical Systems are defined as systems whose failure can generate unacceptable consequences, such as loss of human life. There are in the center of modern infrastructure : commercial aviation, military equipment, autonomous transportation (or autonomous vehicles), and medical devices. Some historical failures, such as Ariane 5 V88 fly or like Patriot missile incident, have painfully reminded us about the necessity of a absolute rigour.

Face of the raise of complexity requirements and connectivity in always growth, the traditional development methods are no longer sufficient. International standards, such as ISO 26262 for the automotive sector, impose the use of rigorous methods to guarantee the safety. It's for that many researchers have worked on this safety methods and have shared their discoveries in papers.

The objective of my internship was to synthetize all these discoveries on the conception and verification of SCS, focusing particularly the formal methods.

2 Description of research

The first week of the internship, we devoted ourselves to discovering the subject. We browsed many articles to have an overview of the subject without going into detail. During this phase, we divided the study in two parts. The first was dedicated to the the life-cycle of SCS and the other was a deep study of formal methods.

Firstly, we analyzed the life-cycle of SCS, we identified two main processes : the code-centric approach and the model-driven approach. We dissected each method to highlight key steps : from initial analysis (Hazard Analysis) to the operational maintenance. These two detail processes are illustrated in the following figure.

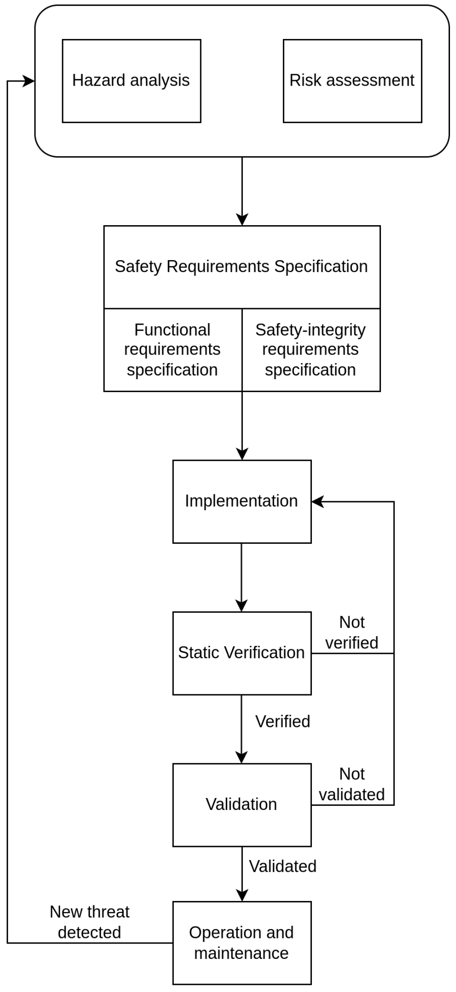
Hazard analysis
Risk assessment
Safety Requirements Specification
Functional requirements specification
Safety-integrity requirements specification
Implementation
Static Verification
Not verified
Verified
Validation
Not validated
Validated
New threat detected
Operation and maintenance

Fig. 1: Code centric process

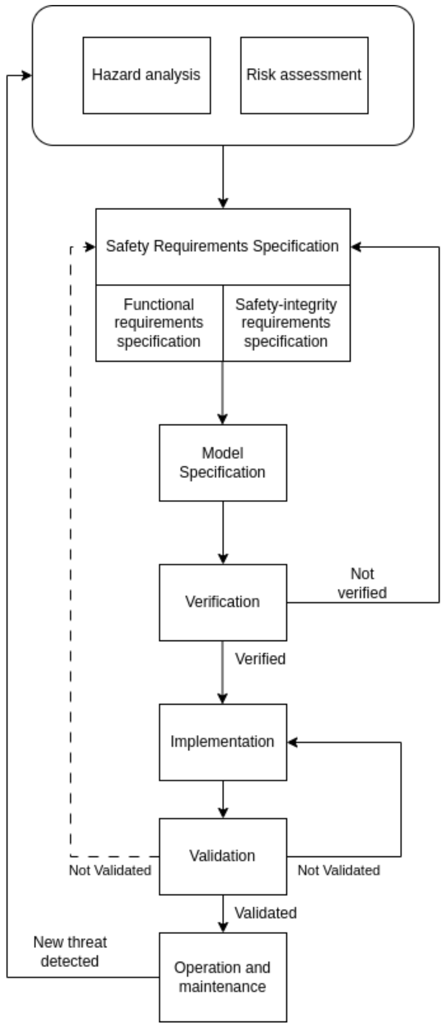

Fig. 2: Model driven process

After that, we studied each step of the life-cycle. We decided to start from the lower-level : validation and implementation, because they seemed simpler. In these steps, it is matter of programmation to create et test the final project. Consequently, there exists a wide variety of programming languages, each with its own features. In the following table, you can see non-exhaustive list of programming language used for SCS.

Programming Language	Language Paradigms	Application Domain	Use Example
Lustre (often through the environment development Scade) [?]	Synchronous, Declarative, Reactive, data-flow	Avionics, Nuclear Power plants, Transportation, Automotive [?]	Airbus A340-600, A380, Eurocopter [?], Schneider Electric: SPIN [?]
Esterel [?]	Synchronous, Imperative, Reactive	Avionic	Dassault Aviation: landing computer [?]
C [?, ?]	Imperative	Avionics / Rail / Water / Energy /Highway / Oil [?]	F35 Jet Fighter [?]
C++ [?]	Imperative, Object-oriented	Avionics / Rail / Water / Energy /Highway / Oil [?]	Safety and Performance Analysis Code for Nuclear Power Plants (SPACE) [?]
Ada [?, ?]	Imperative	Railway, Avionics, Spatial, Healthcare	Eurofighter Typhoon [?], Zephyr light launcher [?]
SPARK [?, ?]	Imperative	Spatial,Healthcare,Avionics	Zephyr light launcher [?], Masten's rocket [?]
Pascal [?]	Imperative	Compiler & Operating Systems	FP-RTOS [?]
Modula-2 [?]	Imperative	Railway, Compiler	urban transportation system in Paris [?]
Java [?, ?]	Object-oriented	Avionics	ScanEagle UAV [?]
Haskell [?]	Functional	Image Analysis	SADLI : Safety assurance in diagnostic laboratory imaging [?]

Table 1: Developping Languages

Then, we proceeded of a deep study of the formal methods. These methods use mathematics for the specification and verification steps and our analysis was divided into two sections.

We focused on the modelling and specification phase. We examined and compared various specification languages, from graphical to textual language. The aim was to understand how their capture the different aspects of a system : its structure, performance or timing. A summary table of existing languages is presented below.

Specification language	Characteristics	Specification Paradigm
Z	– All expressions are typed – No support of concurrency	– First-order-logic – Sequential-oriented [?]
LOTOS	– based on process algebra – represents sequence of event – supports concurrency	Event-based
WhyML	– first order limitation – use of polymorphic types, pattern matching, and inductive predicates – use of record types with mutable fields, type invariants, and ghost code	– first-order logic [?]
B	– supports development of programming language code from specifications – use of set theory – use of refinement – low-level	– First-order-logic – Object-based
TLA	– supports concurrency	Sequential-oriented
Petri nets	– Possibility of representing time – Behavioural specification – Graphical language – Very long conception time if the system is complex – Too much states implies incapabilities to compute and testing all possibilities	– Sequential-oriented – Action-oriented
Timed Automata	– Graphical specification – Behavioural specification – Time verification	Sequential-oriented
UML	– Structural description – Graphical specification – No time notion – No informations about the sequence of actions	Model-oriented [?]

Table 2: Specification Languages

Next, we studied formal verification. We compared the two main families that exist. On the one hand, Model Checking which automatically explores the state space of a model but faces the problem of combinatorial explosion. On the other hand, Proof-based Checking which utilizes proof assistants to demonstrate theorems about the system, offering a stronger guarantee but requiring human expertise.

3 Feedback analysis

The regular feedback from our supervisor was extremely formative. These exchanges refined our methodology for analyzing scientific articles and guided the evolution of our writing.

The initial versions of the report were very descriptive, and the feedback allowed us to add a critical analysis of the methods and employed technologies. The supervisor also praised our autonomy, particularly regarding the integration of the AI issue, although it was not included in the original topic. She also highlighted that the synthetic presentation of tools in the form of tables, it facilitates understanding.

This feedback helped me develop my critical thinking, as I initially tended to focus on the 'catalog' aspect, listing everything that exists, whereas it was necessary to confront the tools to understand why and where they are used. Furthermore, reading numerous scientific articles in English accelerated my reading speed. The feedback highlighted the crucial importance of precise and unambiguous technical vocabulary for effective communication in this field.

4 Conclusion

This papers established a state-of-the-art of the Safety-Critical Systems (SCS) life cycle, it confirms the requirement for a rigorous and multi-step development process . We reaffirmed the crucial role of formal methods for the mathematical specification and verification of complex systems, it's helping to drastically reduce the risk of catastrophic failures. While verification tools continue to mature, the integration of emerging technologies, notably Artificial Intelligence (AI), poses major certification challenges.

The comparative analyses confirm the importance of requirements analysis in the selection of solutions. Our tables highlight the diversity of available technologies, necessitating deep knowledge to select the most relevant tool based on key criteria such as expected performance, learning effort, and other specific constraints.