Disruptive Innovations for the Development and the Deployment of Fault-Free Software

Thierry Lecomte

CLEARSY, 320 avenue Archimède, Aix en Provence, France
thierry.lecomte@clearsy.com

Abstract. Developing safety critical systems is a very difficult task. Such systems require talented engineers, strong experience and dedication when designing the safety principles of these systems. Indeed it should be demonstrated that no failure or combination of failures may lead to a catastrophic situation where people could be injured or could die because of that system. This article presents disruptive technologies that reduce the effort to develop such systems by providing integrated building blocks easier to use.

Keywords: formal methods, safety critical, software development

1 Introduction

Developing safety critical systems [2] requires higher costs (advanced engineering, redundant architecture for SIL3/4[1] applications, extensive verification & validation, specific hardware, etc.). Using off-the-shelf hardware is often a nightmare as it requires to constraint the development cycle by using specific modeling and verification tools, software libraries (and imposed basic operators), etc. Lack of flexibility, requirement for expert practitioners, and resulting higher selling price could dramatically lower competitiveness. Even worse, from a societal point of view, means to improve population safety might not be put into existence because of economical reasons (costs).

To overcome this situation, a solution based on formal methods, the CLEARSY Safety Platform [3], has been designed to ease the development and the deployment of a function, by providing means to model the software, to prove its behavior regarding its specification, to verify mathematically the soundness and the correctness of its parameters, and to ensure a safe execution on a dedicated hardware.

The CLEARSY Safety Platform is made of an integrated software development environment (IDE) and a hardware platform that natively integrates safety principles. Hence the developer only has to focus on the functional design while mathematical proof replaces unit and integration testing. There is no need

[1] Safety Integrity Level. 4 is the highest level, corresponding to a maximum of one catastrophic failure every hundred centuries

for independent software development teams: redundant software is automatically produced from a single model. A certification kit allows the developer to build a safety case and as such ease the certification process. The final safety demonstration remains the responsibility of the developer.

2 Software Modeling

The B formal method is at the core of the software development process (fig 1). Mathematical proofs ensure that the software complies with its specification and guarantees the absence of programming errors while avoiding unit testing and integration testing.

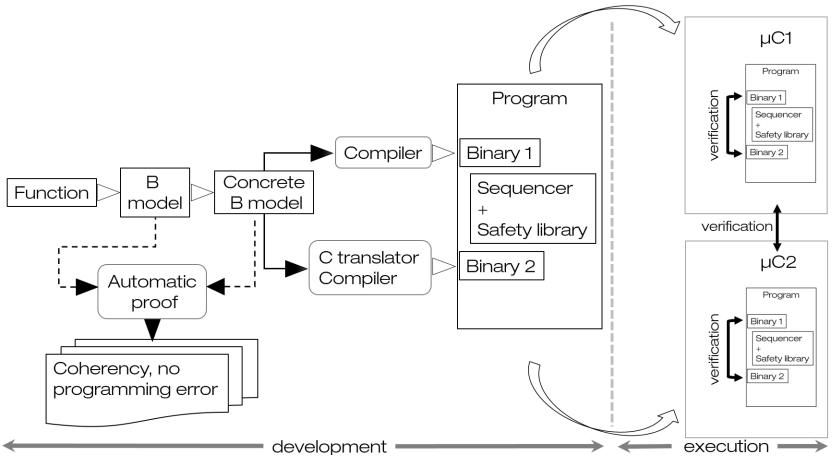

Fig. 1: Software development and deployment : a single formal model is used to generate diverse software. A sequencer and verification functions, developed with B, constitute the safety belt. The sequencer calls the software function every cycle while the verification functions ensure that any divergent behavior is detected and leads to reboot.

Moreover only one functional model is used to produce automatically the redundant software, avoiding the need to have two independent teams for its development. The formal model may be developed manually or be the by-product of a translation from a Domain Specific Language to B (BXml open API). A subset of the B formal language is supported to develop control-command applications. A software project is a B project already filled with an interface description with the board (inputs, outputs, current time), the supported types and operators. The developer only has to fill in the specification and design of the function *user_logic*. In its current form, the software integrated development environment allows developing cyclic applications (read inputs and current time, perform computations, command outputs), run directly on the hardware

without any underlying operating system. There is no predefined cycle time to comply with: the application is run as fast as possible and the time information is managed directly by the application software. With PIC32 micro-controllers, the platform offers up to 100 MIPS for lightweight applications handling Boolean and integer data.

3 Low Cost High Integrity Platform

Software is not safe by itself as it is necessary to consider the processor (and its failures) that is going to execute it (fig 2).

The safety principles are built-in, both at software level and at hardware level (2oo2 hardware, 4oo4 software).

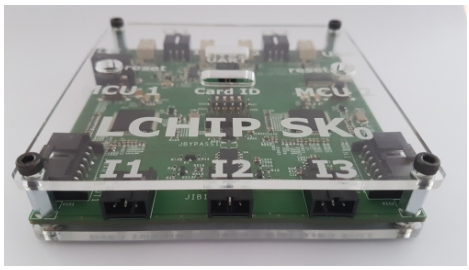

Fig. 2: The CLEARSY Safety Platform: the starter kit SK0

The functional correctness is ensured by mathematical proof. The software is also demonstrated to be programming error free.

The detection of any divergent behavior among the two processors and the four instances of the software is handled by the platform. The safety verifications include cross checks between software instances and between micro-controllers, memory integrity, micro-controller instruction checker, etc. In case of failure, the execution platform reboots. As the outputs require the two micro-controllers to be alive and running, it is not possible to have the outputs commanded in a permissive state in case of divergent behavior. The safety principles of the platform are out of reach of the developer who cannot alter them.

4 Formal data Validation

Software applications are usually developed and validated independently from the parameters or constant data that fine-tune their behavior. In order to avoid a new compilation if the data are modified but not the software, two different processes define the software and the data validations. Data validation consists in checking a heterogeneous data collection against a set of properties / rules

issued from diverse sources like technical constraints, regulation, exploitation constraints, etc. Manual data validation used to be entirely human, error-prone activities.

Formal data validation [1] is the natural evolution of this human-based process into a more secure one where the properties / rules are formalized, to constitute a formal data model (mathematical, based on the B language). It is built from natural language inputs. The verification of conformance between the data collection and the formal data model is performed by a formal tool, i.e. a model-checker (or by a combination of redundant formal tool if required). In the case of the applications developed with the CLEARSY Safety Platform, the parameters of the application but also the technical parameters of the implementation are formally verified.

5 Conclusion and Perspectives

The CLEARSY Safety Platform is aimed at easing the development and the deployment of safety critical applications, up to SIL4. It relies on the smart integration of formal methods, redundant code generation and compilation, and a hardware platform that ensures a safe execution of the software.

A starter kit SK0, developed with the support of the French R & D project FUI LCHIP (Low Cost High Integrity Platform), is available since Q4 2017 for experiment and education.

The building blocks of this technology have already been certified (SIL3/SIL4) in several railway projects worldwide, in particular the COPPILOT systems open and closing platform screen doors in São Paulo and Stockholm.

Together with the full automation of the proof, the connection with Domain Specific Languages, like Grafcet or Structured Text, is expected during this year 2018 to enable the development of functions independently of the B formal language, hence to ease the adoption of these new technologies in the industry.

References

1. Falampin, J., Le-Dang, H., Leuschel, M., Mokrani, M., Plagge, D.: Improving railway data validation with prob. In: Romanovsky, A., Thomas, M. (eds.) Industrial Deployment of System Engineering Methods, pp. 27–43. Springer (2013)
2. Lecomte, T.: Applying a formal method in industry: A 15-year trajectory. In: Alpuente, M., Cook, B., Joubert, C. (eds.) Formal Methods for Industrial Critical Systems, 14th Int'l Workshop, FMICS 2009, Eindhoven, The Netherlands, November 2-3, 2009. Proc. LNCS, vol. 5825, pp. 26–34. Springer (2009)
3. Lecomte, T.: Double cœur et preuve formelle pour automatismes sil4. 8E-Modèles formels/preuves formelles-sûreté du logiciel (2016)