Safety Validation of an Embedded Real-Time System at Hardware-Software Integration Test Environment

Gracy Philip and Meenakshi D'Souza

Abstract As the complexity and functionality of embedded software is increasing steadily, ensuring that their behavior is safe is of primary concern. We propose a Safety Validation Method (SVM) that is used to monitor divergence in safe behavior during integration testing of an embedded system. The proposed monitor observes both application and system level parameters. It can capture effect of potential unsafe scenarios inadvertently created during testing that could recur during actual operation. We present a case study involving the safety validation of advanced fly-by-wire flight control system where the use of the SVM revealed safety critical failures during integration testing.

Keywords Safety validation · Embedded software · Verification · Testing · Monitors

1 Introduction

As the functionality of safety critical systems built these days is complex, safety validation has become a critical part of system development. Powerful cross development environments containing automatic code generators and testing tools are available these days. Commercial of the shelf hardware and software have also become part of systems being developed. In such a development scenario, visibility into the full system has decreased while complexity of the system has increased. Central focus of this paper is on safety assurance of an embedded real-time safety critical system.

G. Philip (✉)
CEMILAC, DRDO, Bangalore 560037, India
e-mail: gracy.philip@cemilac.drdo.in

M. D'Souza
IIIT, Hosur Road, Bangalore 560100, India
e-mail: meenakshi@iiitb.ac.in

R.K. Choudhary et al. (eds.), *Advanced Computing and Communication Technologies*,
Advances in Intelligent Systems and Computing 452,
DOI 10.1007/978-981-10-1023-1_5

Most of the complex functionality of safety critical systems is implemented using software embedded in the system. However, safety is not considered to be a software property alone, it has to be defined at system level [1]. There are two aspects of safety: one is safety critical functionality, failure to achieve it will impact safety. Second aspect is platform-related, introduced by digital implementation in an embedded systems environment, which were not relevant in earlier electro mechanical environment.

Typically there are four main levels of testing: low level testing, software integration testing, hardware software integration testing, systems integration testing. In all these levels, requirement based testing is emphasized because this strategy has been found to be most effective for revealing errors [2]. Each level carries out both normal mode and robustness tests. Each level has its own granularity for fault simulation and result analysis to identify faults. A major lacuna with this compartmentalized testing strategy is that expected results are generated and results are analyzed at that level only, against test objectives at that level. Testing at these discrete levels fail to provide any concrete evidence about safety. Individual test results cannot be stitched together to get a clear picture on system safety. End result is that even after successful tests at all levels with adequate coverage; there can be faults left, hampering system safety.

We propose a safety validation method (SVM) that deploys a safety monitor to expose the vulnerabilities hampering safety. Such a monitor is developed using existing validation environments used for requirements level testing as per D0-178B [2]. There is no extra load on the system under test. We illustrate the working of such a safety validation using a case study involving safety assurance of a flight control system. As per [3], run time verification that deals with the application of formal verification techniques that allow checking whether a run of a system under scrutiny satisfies or violates a given correctness property. Our SVM combines run-time verification with testing in the integration testing environment, with safety criteria as the correctness property.

The paper is organized as follows: Sect. 2 describes the current safety assurance process followed by proposed SVM. Section 3 describes our case study in which the proposed method is applied towards safety validation of digital fly-by-wire flight control system of a modern fighter aircraft. Related work is presented in Sects. 4 and 5 concludes the paper.

2 Safety Assurance and Validation

In this section, we summarize the state-of-the-art processes followed during the development and life cycle of a safety critical system to ensure that it is safe. The term "safety" includes all aspects of software that generically mean that "nothing bad (in terms of behavior of the system) will happen". Current safety assurance methods are based on several stringent development processes and independent verification and validation methods as per DO-178B.

2.1 Safety Validation Method

Typically, hardware software integration testing is the phase in which requirements are validated. In this phase, software is installed in the platform and the integrated system is tested for meeting its requirements. Such a testing involves normal behavior of the system and fault mode behavior. In normal mode testing, a tester focuses on creating test cases to validate each requirement towards checking if the latter has been implemented properly. Fault mode testing is conducted by simulating the faults. This way, adequate coverage is achieved for safety critical software, as mandated by various standards [2].

While such an exhaustive integration testing validates individual requirements, it might fail to capture safety violations that occur at the system level. A tester who focuses on validating one particular requirement may ignore the system level behaviors, some of which could violate safety.

We propose a Safety Validation Method (SVM) to address this issue and capture unsafe behavior of the system during the integration testing phase. In our SVM, safety parameters are extracted through system safety and software safety analysis as shown in the Fig. 1. Figure 1 also describes the SVM, details of which are given in the subsequent sections. Behaviors violating safety are found automatically in real time by observing the values of the parameters as the application executes in the test environment.

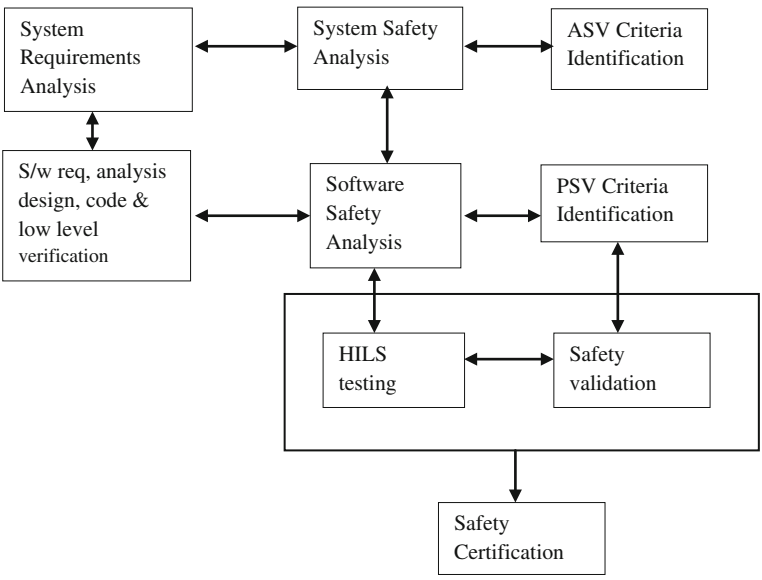

Fig. 1 Safety validation method

2.2 Safety Validation Types

Safety validation can be categorized into two types based on the aspects of safety that are being addressed: Application domain aware safety validation, and platform (system) aware safety validation.

Application-Aware Safety Validation

Application aware Safety Validation (ASV) takes care of safety captured as a part of the overall system functionality. Parameters to be monitored here include all application level data that capture the behavior of the system as running in the system simulated environment. System parameters indicating safety are to be arrived based on system specification analysis along with preliminary system safety analysis. System safety analysis brings out inter dependent sub-systems and their boundaries for achieving system functionality in a safe manner.

Platform-Aware Safety Validation

Platform aware Safety Validation (PSV) takes care of safety interpreted as a non-functional requirement. PSV validates additional hazards brought into the system due to digital implementation like channel failures, watchdog time out, unexpected exceptions and interrupts etc. Safety Monitor captures ASV and PSV parameters in real time, while HSI test scripts are run for normal and robustness test cases.

3 Case Study: Flight Control System

We now present a case study of safety validation of a digital fly-by-wire Flight Control System (FCS) using the SVM integrated with an existing test environment. Using the SVM, unexpected system failures could be captured, which would have occurred during flight operations if left undetected.

3.1 Overview of FCS

The FCS (refer to [4] for an example) is a quadruplex digital fly-by-wire system, it has four identical processing channels and quadruplex input sensors. All four channels process control laws for flight control with identical inputs. Channels work in frames-synchronous manner. FCS is designed to tolerate up to two critical failures. It is interfaced with one air data system per channel and has to control four primary actuators and two secondary actuators. It takes inputs from various sensors from pilot stick and other cockpit interfaces. It also gives out data for pilot displays, crash data recorder and get U home system.

The FCS has four identical computing channels; identical software is loaded in each channel. All the six actuators will continue to receive commands even in the case of failure of any two of the channels. Even when there is quadruplex redundancy available for hardware there is no redundancy in case of software, towards minimizing complexity in design and validation. So it is essential that the software is almost 100 % fault free, towards safe and reliable flight control. The FCS software is written in Ada programming language with SPARK Ada as the coding standard. Control laws were designed in Matlab [5] and Simulink [6] and converted into Ada programming language using Beacon code generator [7].

3.2 Platform-Aware Safety Validation of FCS

We now summarize the testing efforts that were undertaken for FCS. Testability was one of the main design features. Various levels of testing were planned with different test teams—unit testing, software-software integration testing, non-real time testing of control law packages, hardware-software integration testing, systems of system integration testing, pilot-in-loop testing and aircraft integration testing.

Each level of testing had its own objectives and coverage criteria and criterion for pass or fail of the test was limited to analysis at that level. A platform aware safety validation was designed at hardware software integration test level to validate system behaviors in terms of parameters that can be monitored. Criteria for passing a test were amended to include a no fault reply based on SVM along with the specified criteria.

3.3 Integration of PSV in Test Environment of FCS

During integration testing, the software is loaded as a part of the system that runs on an embedded platform containing all the required hardware. Sensor inputs and actuator outputs are simulated as in real-life conditions. Such a test environment is called Hardware Software Integration Test Environment (HSITE). HSITE is capable of fault simulation, and can access all processor and memory variables, along with sensors, actuators, flight control panel, air-data system and display system. It also has the capability to temporarily halt the embedded system processor and read any registers and memory variables.

Test script language provided by the HSITE enables writing of test cases and procedures. Test scripts are written corresponding to each requirement, test inputs generated and passed to the System under Test (SUT) and outputs are recorded. The following psuedocode provides details of how PSV was integrated with HSITE for FCS.

```
for  requirements i = 1 to n do
        {
         power on rest of the system and initialize system parameters
         for test scripts j=1 to m do { // PSV monitoring
            initialize PSV & clear fault records
            run test scripts for each j
            compare results with expected output
    read PSV and fault records & compare results with expected output
         }
        }
```

Our platform aware SVM was implemented using the scripting language provided by the HSITE. Our SVM was able to configure, initialize and load the test scripts, record test outputs and simultaneously track all parameters to be monitored. The validation method was implemented once and designed to work as a stand-alone entity with several test scripts being run one after the other for validating their respective requirements. The monitor had features to clear all faults of the previous run and verify the integrity of each build.

The recorded data in the environment gives vital clues for analysis. The following embedded processor and system interrupts were monitored automatically for FCS: watch dog timer status, frame timer, arithmetic faults, floating point, constraint fault, events and machine fault, Mil bus interrupt, RS422 interrupt, real-time extensions, channels status, average and peak frame time.

3.3.1 Analysis of Results

Our safety validation revealed unexpected exceptions and interrupts which were not handled. We describe computing channel failures detection in detail as it is a safety-critical failure and later resulted in adding new requirements to the FCS.

Analysis of the data provided by SVM helped to observe a channel failure while conducting testing to prove some other requirement (whose test case passed) and this lead to a detailed analysis. The monitor recorded repeated occurrence of real time extension interrupt 74 leading to computing channel failure while running test scripts for proving requirements. As per design this interrupt was expected only when the frame time exceeds 12.5 ms causing the watch dog to time out. But on checking the frame recorded in the monitor it was found that none of the real time frames exceeded the 12.5 ms limit. Consequently, it was understood that triggering of real time overrun was spurious. This resulted in a new requirement to be added—the new one specified a method to distinguish between spurious and real interrupts, with the understanding that if an interrupt is real it will get repeated. For further analysis a derived requirements got added. The SVM could capture deviations in SUT configurations due to patches left behind by testers used for fault simulation. It also helped to capture peak execution time under multiple simulated failure conditions.

4 Related Work

Several researchers and practitioners working with safety critical systems have emphasized the need for exclusive practices, techniques and tools for safety analysis and assurance [8, 9, 1, 10]. In [11] the authors identify safety assurance parameters and techniques to be used throughout the development life cycle and elaborate on the need for independent verification and validation in all phases of the development life cycle. In [11], safety analysis of automated requirement models and their validation is discussed. The work presented here can be thought of as one concrete realization towards safety assurance at the integration testing phase. The proposed SVM not only provides safety assurance during requirements validation but can be continuously used in the later phases of development including safety assurance of incremental releases.

There is a large body of work on run-time verification [3] in the formal verification community. Our work is a case study in run-time verification but doesn't use any formal methods. We just do system level run-time testing, with specifications being given without using any formal notations. In [8] requirements based safety validation that takes an approach inclusive of both static and dynamic analyses for safety assurance is presented. Safety validation follows the requirements based dynamic analysis and testing. Here, we do automatic monitoring of all safety critical parameters, as a part of existing validation environment, combining run-time verification [3] and requirements based testing [8].

5 Conclusion and Future Work

The safety validation method is used for monitoring safety aspects of the system in the system integration and simulation environment, to analyze and perfect the system before its actual operational use. The case study involving application of safety validation to an FCS proved the concept of system aware safety validation, and its ability to catch safety critical errors in real time embedded environment. It was a partial implementation for platform aware SVM, where application aware safety parameters were not monitored. We are currently extending the safety validation to track all application aware parameters. We are also devising metrics to measure the efficiency of the safety validation.

References

1. Nancy, G.: Leveson. Systems Thinking Applied to Safety. MIT Press, Engineering a Safer World (2012)
2. RTCA Software Considerations in Airborne Systems and Equipment Certification DO-178B (1992)

3. Leucker, M., Schallhart, C.: A brief account of run time verification. J. Logic Algebraic Program. **78**, 293–303 (2009)
4. Official website of Tejas. http://www.tejas.gov.in/technology/fly_by_wire.html
5. Matlab. http://www.mathworks/in/products/matlab/
6. Simulink. http://www.mathworks.in/products/simulink/
7. Beacon coder. http://www.adi.com/products/b4s
8. Bhansali, P.V.: Software safety: current status and future directions. ACM SIGSOFT Softw. Eng. Notes **30**(1) (2005)
9. Hunter, B.: Assuring separation of safety and non-safety related systems. In: Proceedings of 11th Australian Workshop on Safety Critical Systems and Software (SCS) (2006)
10. Leveson, N., Alfaro, L., Alvarado, C., Brown, M., Hunt, E.B., Jaffe, M., Joslyn, S., Pinnel, D., Reese, J., Samarziya, J., Sandys, S., Shaw, A., Zabinsky, Z.: Demonstration of a Safety Analysis on a Complex System. Presented at the Software Engineering Laboratory Workshop, NASA Goddard (1997)
11. Modugno, F., Leveson, N., Reese, J.D., Partridge, K., Sandys, S.D.: Integrated Safety Analysis of Requirements Specifications, in Requirements Engineering (1997)