Design and Verification towards Safety and Determinacy in Airborne Software

(Extended Abstract)

Hong Ye
AVIC Xi'an Aeronautics Computing Technique Research Institute
Xi'an, China
mr_yehong@163.com

***Abstract*—This paper focuses on the deficiency of safety and determinacy, due to rapid expansion of the scale, in airborne software equipped on future avionics. Through model-based analysis on engineering defect cases in perspectives of software process and design, we seek for methods combing design and verification that guarantee safety and determinacy in airborne software, thus facilitating research in safety-critical software design.**

***Index Terms*—airborne software, safety, determinacy, design, verification**

I. Introduction

The term *avionics*, being coined as a portmanteau of "aviation electronics", is the science of electronics used in aviation. It covers electronic systems equipped on aircrafts that guarantee accomplishment of scheduled missions while sticking to various specifications on performance. Typical avionics are composed of systems for communications, navigation, display management, etc.. The history of avionics spans over four progressive phases, namely the *independent*, the *federated*, the *integrated*, and the *highly-integrated* architecture, in which the latter two are state-of-the-art avionic architectures requisite for new aircrafts. The F-22 and F-35 are typical aircrafts for integrated and highly-integrated avionics respectively.

In recent years, the term *Integrated Modularize Avionics* (IMA) refers to the integration feature of avionics, and serves as the representative structure of information fusion and resource sharing in avionics. As defined in RTCA SC-200 by RTCA, formerly known as Radio Technical Commission for Aeronautics, IMA is described as "a shared set of flexible, reusable, and interoperable hardware and software resources that, when integrated, form a platform that provides services, designed and verified to a defined set of requirements, to host applications performing aircraft functions".

The development of avionic technology is significantly driven by the improvement of computer, software and microelectronics, wherein the *airborne computer* and *airborne software* have become the core supporting products in IMA, even surpassing the key components of traditional avionics, e.g., structure, sensors, etc.. Airborne software runs on airborne computers, and according to foreign statistics, 80% of the functionality that leads to the capability upgrade of IMA is performed by the airborne software, and therefore after the integration of avionics, the scale of airborne software has been increasing dramatically, for instance, the F-35 aircraft has 8 million lines of code in its airborne software. This however poses new challenges to the development and verification of airborne software. In a traditional sense, the airborne software should feature high reliability, high safety, strong real-time, and high determinacy, whereas the integration of avionics brings inevitably significant effects to these essential features. Consequently, pursuing techniques ensuring the aforementioned four features in airborne software has become a hot research topic in academia. This paper focuses on the design towards safety and determinacy in airborne software. By investigating practical defect cases that affect safety and determinacy, we identify the core issues in the design of modern airborne software, and then present methods for prevention and verification.

II. Analysis on Defect Cases

Software safety refers to "the application of the disciplines of system safety engineering techniques throughout the software life cycle to ensure that the software takes positive measures to enhance system safety and that errors that could reduce system safety have been eliminated or controlled to an acceptable level of risk", as defined in NASA-STD-8719.13A, "NASA Software Safety Standard". *Software determinacy* states that the resources, behavior and state of the software running in the process of system execution can be preset, and no resource exhaustion, illegal operation or unpredictable state occurs during the system execution. In what follows, we give multiple clusters of engineering defect cases regarding safety and determinacy in airborne software.

A. Software Process Control

Case 1 (Traceability): When installing an integrated display system that displays for the pilot the real-time flying altitude to different types of aircrafts, the lack of adaptation of critical altitude values might cause misjudgment of the pilot and thus a fatal crash.

Case 2 (Robustness): During certain flight state of the aircraft (e.g., takeoff and landing), the occurrence of software exception (e.g., divide-by-zero, access violation, etc.) in the flight control system brings immediate threat to the aircraft safety, if no emergency measure is designed.

B. Completeness of Requirement

Case 3 (Requirement itemization): In the requirement specification of a fuel control system, there is a large amount of text describing the dual-redundancy control principle of the underlying system, yet no strict decomposition on functional requirements, performance requirements, and safety requirements, causing repeated switching of the redundancy after installation, and hence spurious reports of the fault phenomenon.

Case 4 (State completeness): In the "Software States and Modes" section of the requirement specification, the software running states and modes cannot be integrally defined, whilst the triggering-event definition for state transition after the fault occurs is missing. This leads to repeated switching of the redundancy after deployment, followed by false reports on the fault phenomenon.

C. Deterministic Scheduling

Case 5 (Partition scheduling): An Integrated-Core Processing (ICP) system distributes application tasks across multiple partitions, allowing the number of partitions to be used at the limit. During the installation experiment, however, data along the three buses is not completely collected and multiple task timeouts occur, resulting in failure of the system mission.

D. Hardware-Software Adaptation

Case 6 (Data packet): An electromechanical public management system needs to collect data through hundred types of external interfaces, which however has been reporting failure due to a long-standing problem of data packet loss.

Case 7 (Cache consistency): In an ICP system, after upgrading the operating system, the CPU speed is obviously reduced along with incorrect data processing during the system execution. This causes the position deviation of the missile launch in the weapon system.

E. Resource Allocation

Case 8 (Space allocation): In the design of an integrated display system, the software engineer is accustomed to using the dynamic space application method to obtain the storage space required by the task. This results in an occasional black-screen phenomenon during the flight of the aircraft.

Case 9 (Space conflict): In the ground integrated experiment, a certain type of mission system halts after several hours of operation with all tasks in the system being in a stagnant state. After analysis, it is confirmed that the task stack suffers from multiple overflows, causing chaos of the program.

F. Fault Isolation

Case 10 (Partition isolation): Consider an ICP system that uses a partitioned operating system. When designing an exception handling for divide-by-zero, the engineer allows the program to continue execution by increasing the Instruction Pointer (IP) by 4, giving rise to failure reports of all the partitions. Therefore, in the design of exception handlers, the generality requirement should be taken into full account, otherwise the fault spreads inevitably once a design error is involved, and hence an improper handling of a local fault may invalid the entire system.

Through all the 10 typical cases given in 6 classes, we showed the effects of design defects on software safety and determinacy, of which the leading causes are summarized as:

1) Incomplete analysis, non-itemization and lack of traceability of the requirements lay serious hazards for the software design process.
2) Non-strict control of the software design, problems ill-conceived and poor margin analysis lead to inevitable chaotic scheduling during the system operation, thus affecting safety and determinacy of the system.
3) Airborne software is typical embedded software where the hardware-software adaptation plays a key role. The lack of sufficient hardware knowledge of the designer for effectively using the functions provided by the hardware will directly reduce the capabilities of the system.
4) Fault isolation is an effective means to improve reliability and safety of airborne systems. The negligence of fault detection and isolation in the design process will bring unpredictable damage to the system.

In conclusion, for avionics of the IMA architecture, the essential cause of the aforementioned problems is the inadequate overall design of the system. Meanwhile, insufficient verification at each design stage is also a source of the issue.

III. Methods for Ensuring Safety and Determinacy in Airborne Software

In addition to improving the overall capability of the aircraft, the application of integrated avionics has brought about a rapid expansion of the scale of airborne software, making the onboard software much more complex —nearly beyond the scope of manual design and verification— than the original software in terms of software structure, working mechanism and state transitions. Thus new methods and tools are expected to improve reliability and safety of the airborne software. According to statistics, 70% of the airborne software failures are introduced mainly in the early development stage, and therefore, improving the ability in early system design and verification can effectively reduce the failure rate of the software later. Furthermore, we should incorporate new software development methods into the development of airborne software. Currently, commonly used methods include model-based software development, model-based system verification, and formal verification. DO-178B/C has incorporated model-based development and verification (DO-331) as well as formal methods (DO-333) into the airworthiness requirements of airborne software, laying the foundation for improving the safety of IMA airborne software.

DO-178B/C, "Software Considerations in Airborne Systems and Equipment Certification", is a software airworthiness requirement issued by the Federal Aviation Administration (FAA). It proposes different airborne software certifications based on different software safety levels, and has also been a measure of improved airborne software safety design. Based on the key points on safety design indicated in DO-178C, we

will illuminate the safety and determinacy assurances to avoid the previous 6 types of defect cases, and further present a case analysis method based on attribute models that enables detection and correction of software defects in rather an early stage of system (software) design.

We will extend the discussion to the key content of the supplementary standard DO-333 issued by the FAA, which mainly addresses formal methods in airworthiness considerations. We then attempt to demonstrate how to apply formal methods in practical airborne software and what problems can be actually solved by formal methods.

In fact, formal methods can enhance safety and reduce the difficulty of certification for safety-critical software equipped on airborne systems. On the one hand, from an empirical point of view, the use of formal symbols is highly beneficial for capturing requirements. It somehow forces programmers to seek all types of problems, which otherwise will be postponed to the code. On the other hand, the formalization of description allows one to conduct rigorous analysis that checks useful properties such as consistency, deadlock privileges, etc., while ensuring the correctness of high-level requirements or design. We show in this paper that applying formal methods can improve requirements, reduce the introduction of errors, increase error detecting rates, and reduce human efforts. Two examples will be depicted to illustrate the use of formal methods in the development of airborne software in order to enhance safety.

IV. Conclusion and Prospect

This paper focuses on multiple challenges raised in the development of airborne software. Since the integration of modern avionics generally reduces the reliability and safety of airborne software, how to ensure high safety becomes the focus of academic research in recent years. With the rapid development of avionics, airborne software is becoming more complex while new development and verification methods are expected, where the model-based development/verification plus formal methods is an effective approach. Although formal methods have been shown to have great potential for meeting the requirements specified in DO-178C, it currently can only be applied to a part of a rather small target, or to multiple targets tailored for development and verification. Formal methods are not a master key to all the problems in development and verification of airborne software and novel methods are desired in future research.