Abstract

The verification and validation (V&V) process of embedded software for critical applications in a distributed Integrated Modular Avionics (IMA) system is a task of great responsibility, because an undetected error can potentially risk a great number of lives and cause significant material losses. This paper presents the requirements for an instrumentation tool that fulfill the needs of the test activities, and the constraints presented by the limits of the platform and the safety analysis. It offers an alternative to shorten the development cycle by allowing the system and software engineers to execute exploratory tests directly on the IMA system when the simulation tools are not representative.

In collaboration with: Andrea Barp, Emilio Bumachar and Rodrigo Marçal

1. Introduction

The DO-178B [5] defines the software verification as a technical assessment of the results of both the development and verification processes to detect and report errors that may be introduced by the development process. Even if testing alone is not sufficient to prove the absence of errors, it is still an important task that is largely used by the software industry. For that reason developing techniques that could increase the error-detection capability is a relevant research topic [3].

The program instrumentation is a method to acquire runtime information about the software internal attributes during a test execution. This is attained by inserting additional statements (the instruments) into the program code to compute and output those attributes. There are numerous applications for the instrumentation, including structural coverage analysis for test cases, program profiling, execution time measurement, among others [3].

This article will present how the instrumentation can help during the integration of multiple applications with the IMA platform and one another, and which features are necessary for a test tool to accomplish that.

2. System overview

2.1. The integrated modular avionics (IMA)

The increase of the aircraft functions implemented by electronic systems and the advent of more powerful and cheaper microprocessors pressed the industry to define a new kind of architecture that could address the shortcomings of the federated approach. The integrated architectures were developed to accommodate the increasingly complex systems by providing a high-performance computing platform that can host multiple applications on a single processor or on a distributed network of processors [6].

Those are the key characteristics of IMA platforms defined on the DO-297 [6]:

- The platform resources are shared by multiple applications. This could induce undesired couplings between the application, which leads to the second item.
- The applications hosted on the IMA platform should be protected from unintended interactions with other applications, so the platform must provide a robust partitioning of its shared resources and other protections capabilities.
- Also, the hosted applications can only interact with the platform and other hosted applications through well-defined interfaces in order to isolate changes between the platform and the hosted applications. Standard application programming interfaces (API) add portability, modularity and reusability.
- Finally, the applications and the hardware may be designed and modified independently of each other. This
way, the applications can obtain incremental acceptance, and modifications have little to no impact on the rest of the platform.

2.2. Platform architecture

The platform contains the components described in figure 1. The arrows indicate the permissible interactions.

**The application layer.** The hosted applications implement the functional requirements of the embedded system. They will be the target of the instrumentation tool described in this paper. For portability and modularity reasons, they can only interact with the core software layer through an well defined interface specified in the ARINC 653 [1]. The APEX (APplication EXecutive) interface provides several services that includes process management, communication and time management.

This paper will focus on applications that implement periodic processes. They normally consist of a data sampling step, the processing and the output generation. This operation sequence, refered in this paper as execution cycle, must be executed at a fixed frequency with strict temporal restrictions.

There are special cases of hosted applications that implements functions that uses services that are not defined in the APEX interface. Those are called service applications and they implement functions that support other applications, such as system wide management functions or communication facilities.

**The interface layer.** The interface between the core software and the application layer is composed by several application programming interface (API) calls. For the sake of portability, they are limited to standardized interfaces, such as those defined by ARINC 653 [1] and a few number of pre-defined libraries.

**The core software layer.** The core software layer contains a real time operating system implements the ARINC 653 part 1 specification [1], meaning that it provides both time and resource robust partitioning. It also contains several device drivers and board level software components that provide access to the hardware layer. No further explanation is necessary as they are not directly related to the topic of this article.

**The hardware layer.** The hardware is composed by several general processing modules (GPM’s). Each GPM is an independent computing platform. All GPM’s have a network device compatible with the ARINC 664 part 7 (AFDX) network [2] specification. Some of them have also an additional I/O device for discrete signal and ARINC 429 communication. The IMA platform is an asynchronous system, meaning that each GPM scheduling is independent of the others.

3. The instrumentation problem

The certification process for IMA systems is divided into six tasks, each addressing different activities of the development process. The task 3 – IMA system acceptance – aims to demonstrate that the hosted applications integrated with the IMA platform are compliant with their requirements (including performance, redundancy management, and IMA platform interface requirements) and that they satisfy the aircraft level safety and reliability requirements [6]. These requirements are commonly expressed as black-box specifications. A black-box specification maps the sequences of stimuli for a software system to their externally-observable behaviors [4], without enforcing on a particular design. However, in some cases it is justifiable to describe the internal structures of the software application, such as when the person specifying the requirement has a greater knowledge about the design than the one that will actually implement it. The verification of these implementation requirements, in turn, can benefit from an instrumentation method when black-box testing is not sufficient to provide the verification evidences.
A entirely different situation is when an error only manifests itself as a failure on the integrated system. The developers therefore cannot rely on simulations or emulated systems to replicate the failure and analyze it. They should be able to inspect the internal structures of the hosted application while it is executed on the integrated IMA platform. In this case, the instrumentation can be used for debugging purposes.

Even after the root cause of the failure is found, a correction on the code still must be done. The developers may need to experiment with a many different sets of parameters and configurations before specifying the correct values. A instrumentation tool could be used in this context as a rapid prototyping tool, providing means to modify the memory spaces that contain the configuration parameters.

Another useful capability is to stimulate the inner software elements with specific input sequences. This feature could be applied when performing robustness testing. While this is a trivial task when dealing with the external events, generating the inputs for a specific function inside the application is not as straightforward. Even thought the external events do affect those inputs, the inner state of the application may also exert influence, so the creation of black-box test procedures becomes a more complex activity. The instrumentation could ease the test creation task by allowing the test environment to directly overwrite the inputs of the internal structure.

Those four use cases are exemplified in figure 2. The monitoring of internal elements both for certification and for debugging are presented in (1). The parameter $a$ is modified in (2) for rapid prototyping, and (3) shows the injection of input signals on internal structures during a test execution.

All those needs, despite being of completely different natures, can be satisfied by the instrumentability. Therefore, the proposed tool shall provide the following features:

1. Allow an external entity (an external entity does not belong to the IMA system and is connected only during the test procedure, such as an oscilloscope or, as in this case, a common desktop or portable computer) to read the state of internal structures of the embedded software, on every execution cycle, while it is executed on the IMA platform, to comply with the verifiability requirements.

   As the test results produced by IDEA could be used as evidence to the certification authorities, the test environment must guarantee the integrity of the recorded data. The AFDX network does not provide reliable transmission in exchange for the determinism, but the data does not need to be sent to the user in real time. So a reliable transmission protocol must be used on the application level.

   Nevertheless, not only the values of the internal structures are important, but the time instant when they occurred must also be recorded. This poses a challenge, as the IMA platform may be composed by a distributed network of GPM’s and is essentially asynchronous (each GPM has its own independent clock) and, as seen before, the network lacks reliability. Without a time reference that is common to all processes, it is impossible to correlate events that occurred in different GPM’s.

2. Allow an external entity to interactively read the values of an arbitrary set of internal structures of the embedded software, on every execution cycle, while it is executed on the IMA platform, for the debugging purposes.

   This feature is deceivingly similar to the preceding. The use case is not a carefully planned test procedure, but a interactive process where many different variables are analyzed before discovering which ones can provide the needed information. Therefore, IDEA should provide a way to interactively select which elements are monitored in runtime and deliver the instrumentation data back to the user as soon as possible after they were produced. The most important characteristic is not the quality of the monitored data, but the usability.

   The consequence is that a significant bandwidth is
needed during the test and the appropriate hardware resources must be allocated. For safety reasons, the resource consumption must be strictly defined to not exceed the hardware capacity, so the amount of elements that can be monitored must be accordingly limited.

3. Allow an external entity to interactively modify the values of internal structures of the system — constants, lookup tables, and the input and output values of the software units interactively and with reliability.

This feature is useful for both allowing the developers to experiment with different configurations by rapid prototyping, and for the execution of test procedures directly on the internal structures. For rapid prototyping IDEA must provide an interactive interface, where the user can command the overriding of variables in run time, so the commands are sent over the network. But, as the AFDX has not reliable transmission, a packet containing the user command could be eventually lost. This can affect the validity of a test case, because the test environment must ensure that the correct inputs are applied during the test execution. Therefore IDEA provides a confirmation mechanism implemented on the application level, as shown in figure 3.

The figure 3 also shows how the IDEA deals with the injection of wave forms for the execution of test procedures. IDEA must buffer the data of wave form before beginning to overwrite the internal element, so that the element will not starve of input data in the middle of the test.

4. The system should maintain its portable and modular characteristics. Those are the driving factors of the IMA technology, but without a proper design those goals may not be achieved.

One consequence of this goal is the decision to use the avionic bus to transmit the commands and data to the instrumented applications. It will share the AFDX [2] network with other functions. This has the disadvantage of limiting the available bandwidth to other (and more critical) functions, but it was the only practical solution on all the GPM’s used in the development of the prototype. It is important to note that those GPM’s are flight worth grade on commercial airplanes.

A secondary consequence of using the avionics communication devices and protocols is the additional capability to use the instrumentation tool on a flight test. Differently from a laboratory environment, it is not always possible to connect dedicated hardware and wiring on the aircraft for instrumentation purposes.

5. Finally, the use of the instrumentation tool shall not compromise the safety of the IMA system. The DO-178B [5] defines that the verification procedures are
executed on a configuration controlled version of the software, so the impacts of any change on that software afterwards must be known. In this case it is important to consider the instrumentation needs while computing the total resources needed by all hosted applications. They can affect the scheduling, memory requirements and the AFDX network configuration.

As seen before, the tool can cause a significant impact on the resource consumption, affecting the worst case execution time (WCET) of the hosted applications and the memory footprint, therefore a version of the software without the instrumentation can present a different behavior than the test results. One option is to maintain the instrumentation code on the final product, but keep it as deactivated code. As the IMA platform provides robust partitioning, parts of the instrumentation code could be executed on a different partition, diminishing the impact on the WCET and on the memory footprint.

On the other hand, the ability to modify the behavior of a certified software is the most notable threat to the aircraft safety. Thus, there must be strong evidences that the hosted applications behavior can never be modified by the instrumentation tool during a regular flight.

4. Design options

Two solutions using different architectures were considered:

1. developing a device driver that is able to access each process memory space, and an application to use that driver to read and write over the internal structures of processes in another partition, or

2. developing one or more service applications and a specific library API which each hosted application would use to provide access to its internal structures.

The first alternative has the advantage of needing no modifications in the instrumented application code, but still has an important drawback: it is not a portable solution, as the driver is specific for each platform, and it demands a modification in the operating system.

The second option does not need an application with special permissions to bypass the partitioning system, as each instrumented application will provide the services to read and write over its internal structures through the inter-partition communication services defined in ARINC 653 [1]. On the other hand, each instrumented application must contain additional code to implement those services. This problem can be mitigated using a library to encapsulate most of the instrumentation functions.

As the portability is one of the design goals of this tool, the second solution may be the better option.

5. Future developments

A prototype of this tool is currently in development to be used on a flight control system (FCS) on a proof-of-concept rig.

The next step is to develop a production version of the instrumentation tool and apply it to a real aircraft development process.

6. Conclusion

The development of a test environment that complies with the requirements presented in this paper is by no means a small task. Even being portable, an extensive effort in the configuration of the IMA system is still necessary, both for processing power and bandwidth allocation, and for specifying the list of elements that are readable/writable. The former can be automated, but if the test results are used as certification evidence then a special attention on the correctness of that list is necessary. The interlock function is also critical for the safety of the system and it is expected the certification authority to intently focus on this aspect.

On the other hand, it is expected that the finished prototype to already reduce the workload during the system integration. Its main concern today is on system ground tests, but it is envisioned in the future the usage of this tool also on flight tests campaigns.

References


