Comprehensive Validation Platform for Complex Aircraft Avionics Systems

Nicolas, Favarcq[1], Pierre Palluau[1], Olivier Sentier[1], Jia Wang[2]
[1]Cassidian Test & Services, Colomiers, France
[2]EADS China, Beijing, China
{Firstname.Lastname}@cassidian.com, jia.wang@eads-china.com

Abstract - Since 10 years, aerospace industry is facing an unprecedented cycle time reduction driven by high competition between Aircraft manufacturers. To safely sustain such optimizations, virtual test is becoming essential in avionic development cycle. Modern aircraft validation techniques rely heavily on virtual test but integration of this activity in the validation process is still far from being optimal.The goal of this paper is to present a novel approach, in terms of process and technologies, to maximize reuse between virtual testing and more conventional system validation strategies for avionics systems.

Index Terms - Virtual test, model based testing, Verification, validation

I . Introduction.

The efficiency of Integration, Validation and Verification (IVV) process is probably the most important success factor in the development of complex aircraft systems.

An effective IVV process allows detecting mistakes and discrepancies in the design earlier in the development cycle.

As an example, the foreseen production rate for the AIRBUS A30X (A320 successor) is about 30 aircrafts per month, 2 years after the Entry Into Service. At the first flight of the A30X, 100 aircrafts should be in Final Assembly Lane (FAL) [1].

Any corrective operation with such forecasted delivery ramp-up would have heavy consequences in terms of costs and time delays and this impact would be all the more tragic since the correction would occur late in the development cycle.

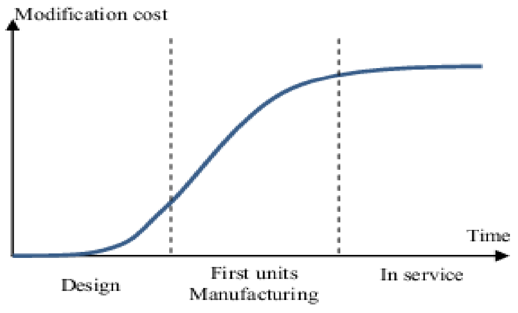

Fig.1 Modification cost vs. Life cycle

Current development methods lead to an experienced 30% rework due to (among other causes) a lack of maturity in systems [2].

Tools and simulation platforms used to perform the IVV activities are a key lead to improve the whole process performance.

II . Avionics IVV overview

Typical validation stages for complex systems (aircrafts, torpedos, etc.) are [3]:

1) **Equipment level testing**: embedded computers are integrated on test benches. Inputs are simulated and calculator's internal parameters are monitored in real time through a dedicated debug link.

2) **System level testing**: the interaction between several computers is tested in an operational environment. This test can be performed on a mock up including real components such as actuators and sensors. Iron birds and flight simulators are perfect example of such kind of platforms.

3) **Test on first prototype** (eg. flight tests) with a "heavy" test instrumentation.

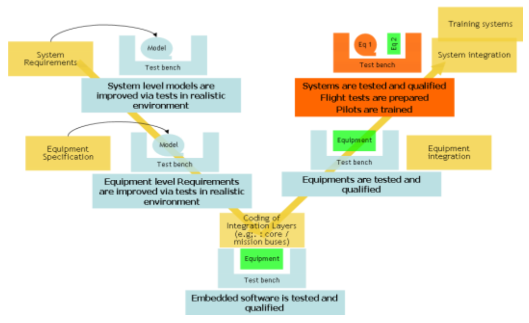
Training systems
System Requirements
Model
Test bench
Eq 1
Eq 2
Test bench
System integration
System level models are improved via tests in realistic environment
Systems are tested and qualified
Flight tests are prepared
Pilots are trained
Equipment Specification
Model
Test bench
Equipment
Test bench
Equipment integration
Equipment level Requirements are improved via tests in realistic environment
Equipments are tested and qualified
Coding of Integration Layers (e.g;. : core / mission buses)
Equipment
Test bench
Embedded software is tested and qualified

Fig. 2 Typical IVV process

IVV tools impact significantly the test activities. These tools must be designed to perform as much tests in the early stages of the development as possible.

To do so, the main functionalities of these tools are:

- Simulate missing components of the system under test. For instance for an aircraft system, in the initial mockup, the inertial navigation system is usually replaced by a six degrees of freedom system simulation.

- Spy and record the communication between the various components.

- Spy and record internal calculator's parameters through dedicated debug links.

To make the IVV tools usable in the early stages of the development cycle, these tools should be completed with virtual testing techniques. Virtual testing consists in replacing embedded aircraft computers by their simulation equivalent. This simulated equipment can be based on the rehosted embedded code or can be a dedicated implementation of control logic for simulation purpose. In such a configuration, aircraft behavior is simulated by a behavior-model often provided by the aircraft manufacturer.

Virtual Test in upstream phases is a way to reduce integration and validation costs [4]:

First, by reducing dependency on physical test means:

- These heavy test means are clearly identified as integration activities bottleneck,

- These heavy test means are much more expensive than their virtual test platforms counter parts,

- Some of these heavy test means (such as iron birds) could be simplified.

Second, by maturing test procedures in early stages of the development cycle, before integration on final integration test means.

III. Virtual and physical test platforms.

A. Example system.

For historical reasons virtual testing and physical testing are executed according to completely different philosophies and on different kinds of test platforms. As a consequence, the potential reuse between virtual and physical testing is so minimal that a real disruption occurs in the IVV process.

To illustrate this, let's take an example: 2 avionics computers {A, B} control a physical system S with 2 sensors $\{s_1 ; s_2\}$ and 1 actuator $\{a_1\}$. We assume that actuators commands and sensors data are refreshed at 1 ms period.

Computer A is a local processing Input/Output node. Its internal loop runs with a period of 1 ms and it sends periodic data frames to B, $M_{A \rightarrow B,}$ every 20 ms.

Computer B is a remote controller. Its internal loop runs with a period of 20 ms.

In a real avionic system, all the components run completely asynchronously [5]: computer A and computer B clock are not synchronized. Thus, sampling of sensors and actuators command is correlated with A but not with B cycles.

Message $M_{A \rightarrow B}$ is synchronized with computer cycle (ie. it is send each 20 cycles of A computer).

For its internal processing B uses the last received $M_{A \rightarrow B}$ message from A. The variable age varies in the range]0ms;20ms[depending on the synchronization status of A versus B clock.

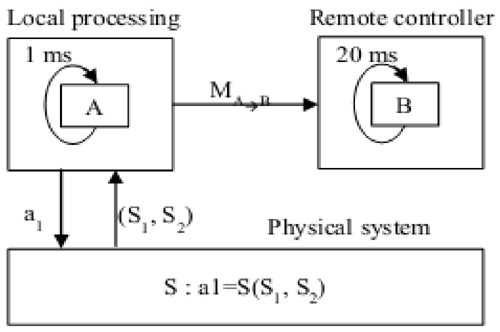

Fig.3 Example system

B. Virtual test approach

Virtual test approach consists in testing the system in very early stages of the development with pure simulation models, which means that A, B and S components are simulated.

Depending on avionic program maturity, A and B computer simulations can be either rehosted code from real computer or simulation explicitly created for virtual test purpose.

S System simulation is very often built with modeling tools such as Matlab® or LMS® but can also be written in more classical programming language such as C language..

The resulting models are integrated in the simulation platform in charge of scheduling all the models execution and to support data exchange between them. The goal is to perform system level testing [6].

The execution scheduler of the virtual test platform focuses on behavior-determinism; time-determinism is not considered as a relevant constraint and is therefore not taken into account. If the simulation is run several times: it must produce exactly the same results. Typically this means that great care is taken to describe accurately, in the simulation platform, the execution order of the different models being part of the simulation. Simulation behavior is often described with framework such as HLA (High Level Architecture) [7]. Behavior determinism is perceived as an essential characteristic because it allows running automatically software non regression test.

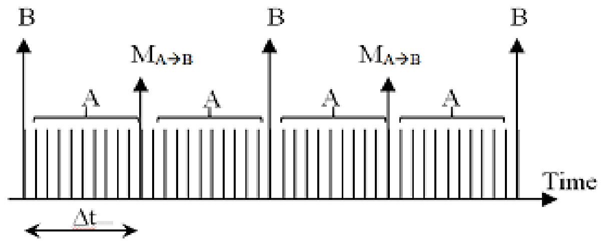

Fig. 4 Models synchronisation

Generally, time resolution is selected at the lowest internal loop cycle of embedded computer- in this example 1 ms. A fix offset is used to represents the asynchronicity between A and B. The consequence is a fixed Δt between B cycle start and $M_{A \rightarrow B}$ message generation by B computer[8].

In pure virtual test systems, time determinism is not the issue. At this stage, there is no link with physical hardware. The only time constraint is to run as fast as possible in order to optimize the test campaign. The time is completely simulated by the scheduler: simulation time is not synchronized with real time. This implementation allows to not care about each model real execution time, next stimulation step is executed when the previous one is finished regardless of the real amount of time spent for the execution.

C. Hardware in the loop test approach

The concept of hardware in the loop test platform consists in integrating real components (avionics computers for instance) in a simulated environment. The goal is to increase the level of test representativeness compared to pure virtual test.

To illustrate this, let’s continue with the example described above assuming that computer B is now simulated.

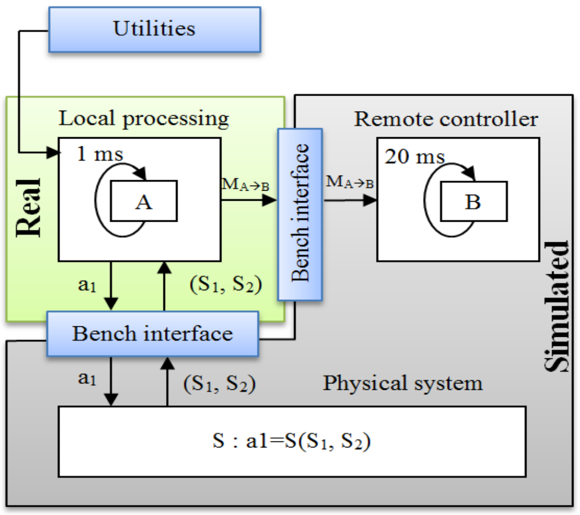

Fig. 5 Hardware in the loop test platform

The test platform has to manage at least the following functionalities:

- Schedule the models B and S (avionics computer and environment respectively).
- Interface B and S model inputs and outputs with real signals sent and received by the real avionic computer A.
- Spy and log the communication between each component,
- Provide utilities to real components (power supplies, cooling, etc.).

Avionic computer A is real. This means that there is no possibility for the simulation platform to control its execution time. This fact has 2 main consequences:

1) Simulation platform must execute B and S model with hard real time constraints to ensure consistency with algorithm execution speed in computer A. This real time constraint has a significant impact on simulation platform architecture. First of all, it must run on a dedicated Operating System (OS) such as VxWorks® , LynxOS® or Linux associated with real time patches. Then, models execution time must fit in the real execution cycle corresponding to the counterpart real computer cycle time (1 ms in the example of computer A). This second point can be extremely challenging and may require having extensive processing power (computer clusters may be an option) [9]

2) Behavior determinism is not achievable anymore. Computer A and simulation platform clocks are not linked (as in real avionic systems). Simulation platform algorithm (simulating computer B) and real computer algorithm run asynchronously. The consequence on the example system is that Δt time between B cycle start and $M_{A\rightarrow B}$ message generation by B computer is not fixed anymore (see fig. 4).

In so many words, if the same test is run twice, the test result will probably not be exactly the same. This corresponds to real system architecture and consequences on system validation consistency have been analyzed in depth [10].

C. *Virtual vs. physical test platform*

We saw that virtual and physical test platform have different targets:

- Deterministic behavior and observability for virtual test.
- Real time constraints for hardware in the loop physical platform. Intrinsic indeterminism was also pointed out.

The consequence of this major difference is that very often, different tools are selected. A disruption in the IVV process is introduced. Reuse of components such as test scripts and simulations from virtual test to physical test is therefore extremely difficult.

The last part of this paper presents a novel approach developed by Cassidian Test & Services (an EADS company). It consists in enhancing an existing hardware in the loop framework named U-TEST to run deterministic virtual test.

III. Linking the 2 approaches.

A. *U-TEST introduction.*

U-TEST is hardware in the loop framework addressing aerospace and defense industry. It was originally designed to support IVV process for Airbus programs (A380, A350) for both aircraft avionics suppliers and the aircraft manufacturer. Its scope has then been extended to other aircrafts, helicopters and programs such as torpedoes.

The functional diagram presented fig. 6 highlights the two main functionalities of this real time core:

- The first one is to transform raw data “flowing” on the system buses (MIL-STD-1553, AFDX, ARINC429, ANALOG, etc.) used between various control units to high level data for it to be understood by a system engineer (e.g. altitude, ground speed, etc.). This functionality corresponds to the bench interface introduced fig. 5.
- The second main functionality of U-TEST is to handle “application-specific” code execution. This covers execution time control and scheduling of the task. Application specific code can be environmental models, system under test calculator simulation, or test procedures. This second functionality requires a hard real time scheduler dedicated to execute models and simulations. For U-TEST, a Linux operating system was selected and extended with the Xenomai patch, dedicated to provide real time performances to final applications.

These two set of functionalities heavily interact through the exchange of system variables. This communication is achieved by a Global Data Space which is a type of shared memory, representing the instantaneous image of all system variables. From a technological point of view, this virtual data space is based on the publish-subscribe paradigm [11].

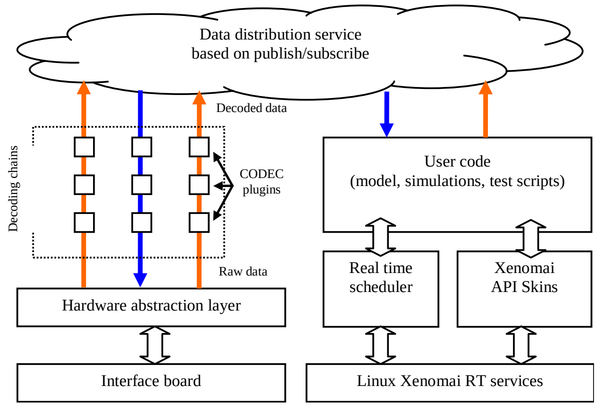
Data distribution service
based on publish/subscribe
Decoded data
Decoding chains
CODEC
plugins
Raw data
Hardware abstraction layer
Interface board
User code
(model, simulations, test scripts)
Real time
scheduler
Xenomai
API Skins
Linux Xenomai RT services

Fig. 6 U-TEST internal architecture

U-TEST real time scheduler relies on OROCOS framework. It is an open source solution intensively used in robotics [12]. Orocos is used to control task execution while providing strong support to control the data flow exchanged between them.

Inside U-TEST, each Orocos task corresponds to a model being part of the simulation (A, B or S in the example introduced fig. 3). Each task is defined as “periodic”: it is executed automatically at each specified period (1 ms in the case of model A for instance). Linux Xenomai patch provides the real time execution stability. An overrun mechanism checks that task execution do not exceed the simulation period.

Orocos by itself does not provide any mechanism to ensure execution behavior determinism as defined in §3. This functionality is simply not in the scope of the framework targeting initially robotics command computers.

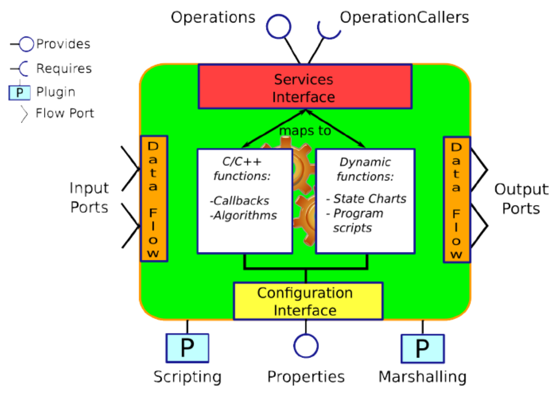

Fig. 7 OROCOS task

B. *Expanding the scope.*

As explained above, U-TEST initially focuses on hardware in the loop physical test framework. In order to offer a consistent offer covering all the IVV activities, Cassidian Test & Services implemented a set of enhancements to efficiently treat the virtual test problem. The goal is to provide a single tool to maximize reuse of test scenarios and to leverage user training.

The 2 main enhancements allow using efficiently U-TEST as a pure virtual test platform when no real hardware is connected.

Enhancement #1: Capability to deterministically execute simulation. In so many words, be able to describe and control which model is triggered at each simulation step. In the simple example introduced in §3, it consists in having a fixed Δt fig. 3.

The option selected to implement this enhancement is to overload Orocos scheduling class. Orocos standard scheduling model consists in executing asynchronously each periodic task at a specified rate. Individual model execution triggering is managed by real time OS timer to guarantee time execution accuracy. However, no mechanism is implemented to ensure time synchronization of models together.

Orocos scheduling class overloading consists in adding a layer between the real time OS timer and the model execution triggering mechanism. This layer is trigged at the least common multiple (LCM) of models execution periods (1 ms in our example). It implements a cyclic state machine to trig model execution at the specified rate. Model execution order is described in XML format to allow both human editing and computer generation.

It generates very low execution overhead and has no significant impact on real time performance. In addition this solution provides the capability to generate complex scheduling scheme of the task.

This solution allows benefiting from Orocos infrastructure (inter task communication for instance) while having a fully behavior-deterministic scheduler.

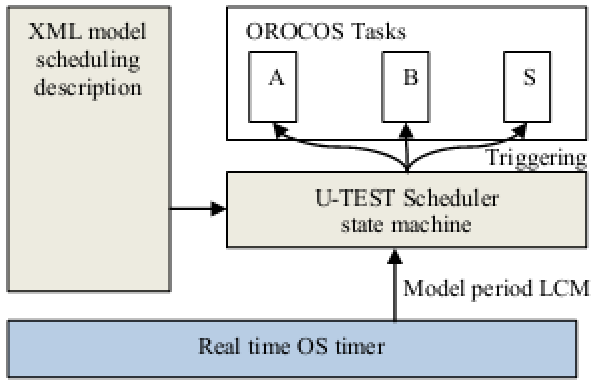

Fig. 8 U-TEST State machine

Enhancement #2: Simulated time capabilities. Originally, U-TEST execution time was correlated with real time because Orocos time base is linked with real time operating system. In case no real hardware is present, there is no need to have this constraint and real time becomes a disadvantage:

- If the simulation is lightweight, system could have run faster than real time and test time is lost (some lightweight simulations can run several dozens of time faster than real time).

- If the simulation is complex it obliges to oversize the execution platform to meet the real time constraints.

The solution implemented to overcome this problem was called “time virtualization”: a simulated clock is implemented between U-TEST system clock and operating system clock. In practice the mechanism can be compared to software Phase Locked Loop (PLL) controlled by the simulation execution Overrun Time Margin(OTM). OTM can be seen as time difference between model real execution time and model specified execution period. When OTM is important (for instance a 10 ms periodic model executes in 1 ms) the virtual clock frequency increases and vice-versa.

In practice simulation is started with very low simulated clock frequency to avoid initial overrun. Simulated clock speed is progressively adjusted (increased) to maximize simulation execution time.

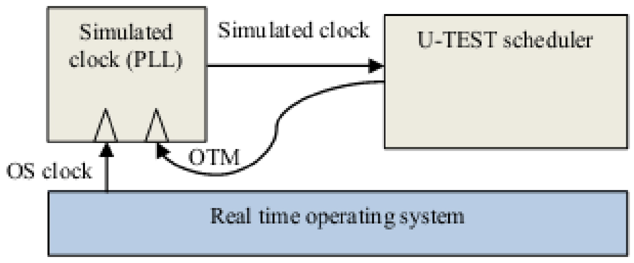

Fig. 9 Simulated clock principle

This mechanism is activated for pure virtual test mode only. If real hardware is in the loop OS real time clock is used.

IV . Conclusion

This paper presented a novel approach to extend a hard real time test platform to perform avionics virtual test.

Cassidian Test&Services demonstrates with U-TEST that both software and hardware technologies are ready to face the challenge of test system unification in the avionic IVV process.

The next steps to ensure a real take off of such technologies are:

- The emergence of standards: this new avionic test strategy needs standardization to ensure exchange of data, models and test cases between the players (System Manufacturers, Equipement Manufacturers, subcontractors)

- The convergence of test processes within the big companies (System and Equipment Manufacturers) with a common purpose:test activities have to be anticipated in early design phases.

References

[1] David Kaminski-Morrow, “A320 successor to be assembled entirely in Hamburg”, Flight Global, August 2009
[2] JC Roussel, “Benefits of Requirement Engineering With Doors”, TELELOGIC capital market event, 2005
[3] D. Briere et P. Traverse, “AIRBUS A320/A330/A340 Electrical Flight Controls - A Family of Fault-Tolerant Systems”, FTCS-23, 1993
[4] Walters, S.A., “Virtual test station (VTS)”, NAECON, 1993
[5] R.Alena, J. Ossenfort, K. Laws, A. Goforth, F. Figueroa, “Communications for Integrated Modular Avionics”, IEEEAC, 2006
[6] D. Monteverde, A. Olivero, S. Yovine, V. Braberman, “VTS-based Specication and Verication of Behavioral Properties of AADL Models”, ACES'08, 2008.
[7] J.-F. Tilman, P. Morignot, J.-C. Poncet, N. Strady, B. Patin, “ Athéna: a Generic Multi-purpose Environment for Simulating Complex Systems”, In Proceedings of the ESIW, 2004
[8] Saghi, G., Reinholtz, K., “High speed simulation scheduler for Cassini spacecraft”, 16th DASC, 1997.
[9] J. Falasco , “Avionics sensor simulation and prototype design workstation using COTS reconfigurable computing technology”, Proc. SPIE 6561, Unmanned Systems Technology, 2007
[10] N. Neogi, “Safety Issues in the Asynchronous Control of Critical Avionics Systems”, DASC, 2011
[11] S. Oh, . Kim, G. Fox, “Real-Time Performance Analysis for Publish/Subscribe Systems”, Technical Report October 20 2008
[12] H. Bruyninckx, “Open Robot Control Software: the OROCOS project”, ICRA, 2001