Simulation-Based Testing of Software in Space Applications

Dr. Sergio Montenegro [1], Prof. Stefan Jähnichen [1], Dr. Olaf Maibaum [2]

[1] TU-Berlin / Fraunhofer FIRST,
[2] Deutsches Zentrum für Luft- und Raumfahrt e.V.
Simulations- und Softwaretechnik
Stefan.jaehnichen@first.fhg.de, sergio@first.fhg.de, Olaf.Maibaum@dlr.de

Abstract: This paper deals with the software-in-the-loop test approach being developed by the consortium project SiLEST (DLR, TU-Berlin, IAV, FhG FIRST, Webdynamix). We present a layer structure of the control loop that allows components of the environment simulation to be used for hardware-in-the-loop and software-in-the-loop testing of embedded systems software. The approach is specifically designed to test software behaviour in disturbed operating conditions, such as in a harsh environment, for example. In space applications, intensive radiation can corrupt computations and stored data. In addition, electronic devices such as sensors age much faster than on earth so that changed sensor deviations must be expected. Much the same is true of numerous other embedded systems, e.g. in automotive applications. Here, too, the electronic components are exposed to extreme conditions (temperature) and are subject to ageing processes.

1. Introduction

The development work was motivated by difficulties encountered in testing the BIRD satellite. BIRD is a technology demonstrator and a space fire alarm (see Figure 1).

Figure 1: BIRD in orbit

The BIRD microsatellite mission demonstrates the technical and programmatic feasibility of combining ambitious science and new – not yet space-proven – advanced technologies under fixed budget constraints. Demonstrating new microsatellite technologies is a key objective of the BIRD mission. The technology experiments demonstrate the limitations and advantages of the newly developed components and technologies. The BIRD microsatellite (mass = 92 kg) was launched with the Indian PSLV-C3 from Shar on 22 October 2001 into a sun-synchronous circular orbit at an altitude of approx. 568 km. BIRD orbits the earth every 90 minutes, scanning a 300 km-wide band. It is capable of detecting, analyzing and reporting fires of 8 sq m in size upwards. BIRD's fire-detecting capabilities exceed the most optimistic expectations. Since its launch, BIRD has functioned with high reliability, despite some radiation-related problems and hardware failures.

Most functions in BIRD are software-controlled using the real-time operating system and middleware BOSS. BOSS was designed for space applications, in particular for dependability, safety and simplicity. Our aim was (and is) to attain the greatest possible dependability of embedded systems by reducing development errors (through simplicity) and handling runtime anomalies (by fault-tolerance support). The principles underlying the construction of BOSS and its middleware were: design and build an irreducibly complex system using modern framework technology for the underlying operating system and component technology for the middleware and its applications. The results are very promising. BOSS has been in continuous use in space (BIRD satellite) and in medical devices for a

number of years now. Even complex functionality can be implemented very easily using BOSS.

Ensuring BIRD's correct functioning was a very difficult task, not only because the functionality is largely implemented in software but also because BIRD makes use of numerous new technologies, making the test phase particularly important and difficult. Some functions are not easy to test on the earth: for instance, attitude control requires zero gravity and the solar sensors need extremely high light intensity. For such cases, a highly complex testbed was built (see figure 2). For future missions, it is planned to reduce this complexity, though not at the expense of reliability. That is why we are building a virtual simulation environment for satellites, which can be used equally well for other embedded systems.

Figure 2: Testbed

2. Testing Adaptive Systems

The environmental conditions in many embedded systems are non-optimal. In space applications, radiation is a problem, while earth-bound systems, e.g. in the automotive sector, are subjected to cycles of heat and cold, moisture and corrosion, vibrations, fouling and mechanical wear, which

cause sensors and actuators to age. The software-implemented control system must take this into account by using adaptive mechanisms to adapt to gradually changing conditions. Such adaptive mechanisms may, for example, involve adapting filters for sensor data or reconfiguring redundant sensor networks, which will be essential for future X-by-wire systems and are customary in space applications to extend the useful life of on-board systems.

Given the variety of ageing processes and failure events affecting sensors and actuators and the high complexity of sensor networks, the testing of these adaptive software mechanisms represents a considerable challenge. SiLEST will incorporate a "software-in-the-loop" (SiL) simulation to test software behaviour in an aged system for the automotive and aerospace application domains. "Software-in-the-loop" testing means that testing of the real software – including all restrictions with regard to resources – is carried out in a simulated environment or using experimental hardware.

Furthermore, the process of developing embedded software systems in a technical environment with high correctness, safety and robustness requirements often accounts for approx. 50-80% of the total development effort in the verification and validation phases [RB02]. Despite this considerable effort for testing software, potentially fatal software errors continue to be common occurrences. By seeking to improve the test process for embedded systems, SiLEST promises to reduce development effort and increase the reliability of embedded software systems.

3. From Real Devices to Software-in-the-Loop

In addition to classical functional and module testing, embedded systems software must also be tested in the complete controller loop including target hardware and target devices (see Figure 3).

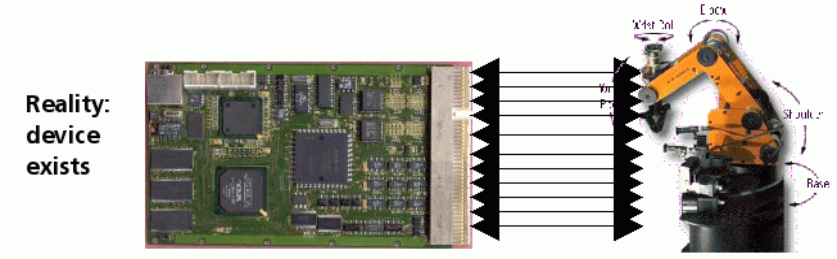

Figure 3: Control software in a real device

Embedded systems software is an integral part of this loop and cannot be considered separately from it. Every output of the software in the loop triggers feedback on the sensor data. This must be taken into account when testing embedded systems software. The conventional approach to testing embedded systems using simulation is the HiL (Hardware-in-the-Loop) test (see Figure 4)

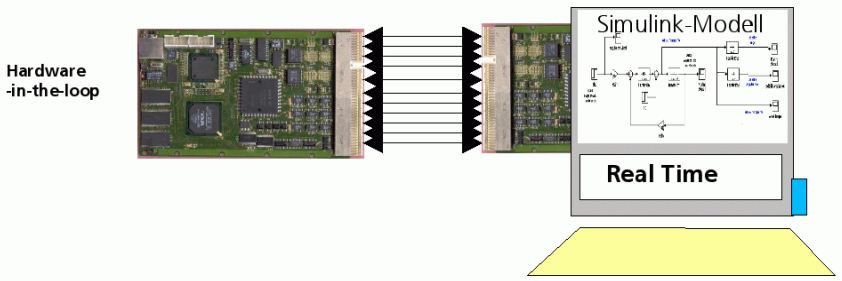

Figure 4: Hardware-in-the-loop

This approach uses laboratory prototypes of the technical system under test or simulations. Testing using a laboratory prototype is highly cost-intensive and can only be carried out at a very late stage in the development process because the relevant system hardware has to be available for testing. Simulation testing involves simulating the environment, the sensors and the actuators of the system being tested. The sensors' and actuators' simulation is coupled via the processor board's hardware interfaces and special VME or PCI interface cards (a complex business).

The HiL test approach has a number of drawbacks. The hardware needed for the test setup is not normally available until late on in the development process, and often then there is only a single specimen of the Hardware Simulator, which is very expensive. This makes the test setup cost-intensive and laboratory-bound. The coupling of the simulation and the embedded system makes it difficult to keep a consistent check on the state of the software and the environment. This, in turn, makes it hard to diagnose errors in the embedded system.

Our alternative to HiL testing is to use a SiL (Software-in-the-Loop) test approach. With this approach, the system under test and the environment simulation are coupled via a single communication connection (e.g. Ethernet) (see Figure 5).

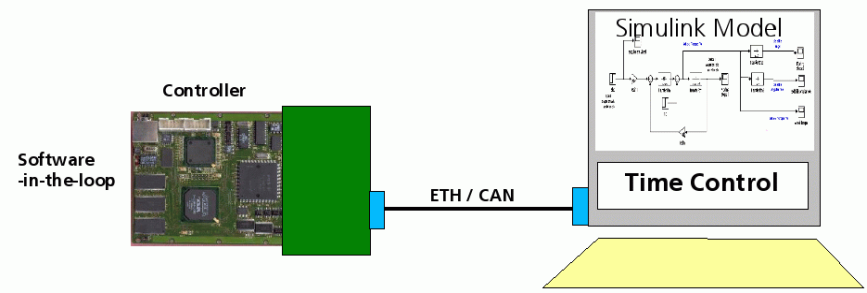

Figure 5: Software-in-the-loop

The SiL test approach enables the in-circuit test harness to be dispensed with and means that the testing is no longer laboratory-bound. And with no need for special hardware, testing can begin early on in the development process. The only essential requirement for testing is a controller approximately corresponding to that in the final product in terms of temporal behaviour and storage capacity and a simulation computer. By dispensing with hardware interfaces, SiL testing allows close coupling of the environment simulation and the system under test. This coupling method enables a consistent state of the simulation and the software under test to be achieved, which can be used for error diagnosis and as an initial state for further test runs. Figure 6 shows the simulation possibilities for different test approaches.

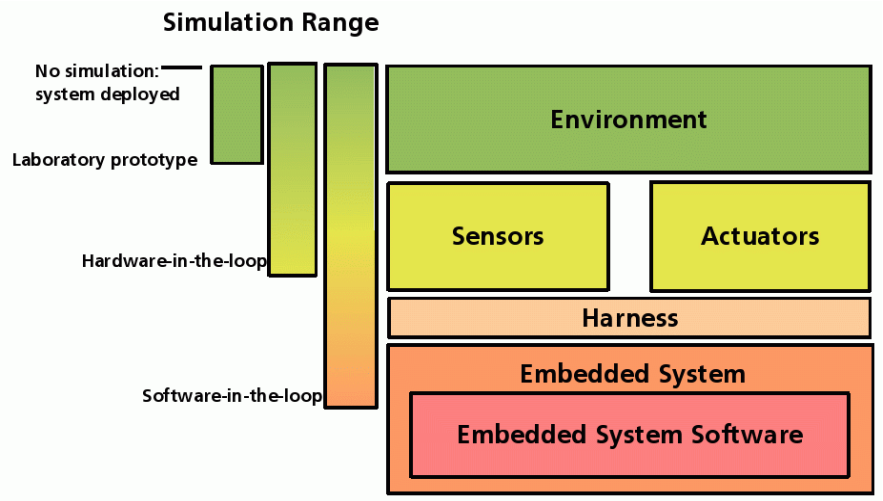

Figure 6: Test approaches

4. Our SiL Approach

We aim to replace the real environment of the embedded software by a simulation, without the embedded software being aware of this (see Figure 7). In a real system, the operating system executes I/O accesses using the corresponding I/O drivers. Our SiL approach dispenses with I/O devices and drivers altogether. Instead, all I/O accesses are converted to UDP messages (a simple Internet protocol), which are sent by Ethernet (ETH) to a remote simulator. The simulator computes the environment reaction and returns the simulated environment conditions, also using UDP messages too.

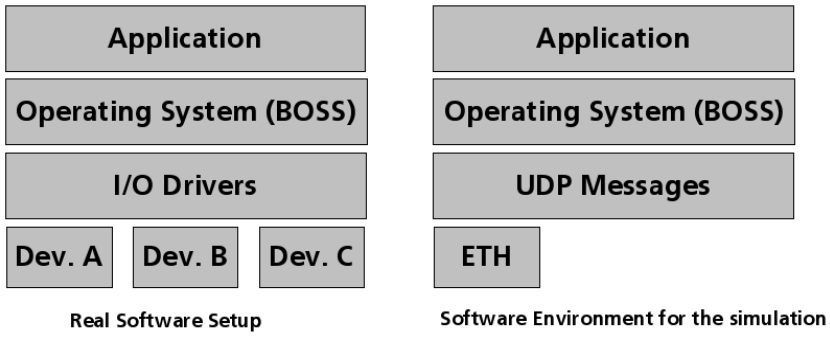

Figure 7: Embedded software for the SiL test

The biggest problem faced here is the system's time behaviour. HiL simulations must run in real time, which makes the simulator highly complex and expensive. Our SiL simulation does not run in real time, but the embedded software is not aware of any time delays, caused by the UDP communication and simulation steps. This is the challenge confronting us.

To the embedded software, the system appears to run in real time. To achieve this, the operating system (BOSS) was extended by the addition of a virtual time manager, which can freeze the time and all software activity while UPD communication and synchronization is under way. This means that the virtual time is not continuous. The system has two notions of time: virtual/simulation time and real time (see Figure 8). The virtual time of the embedded software has to be synchronized with the simulation time (they should be the same) in such a way that to the embedded software the simulation time appears to be real and continuous. By contrast, the (real) real time is that perceived by an external observer. Since the simulation runs slower than the real world, the time of the embedded system runs

faster than the real time. To keep them synchronized, the virtual time has to be stopped every time the embedded software tries to access the external (simulated) world.

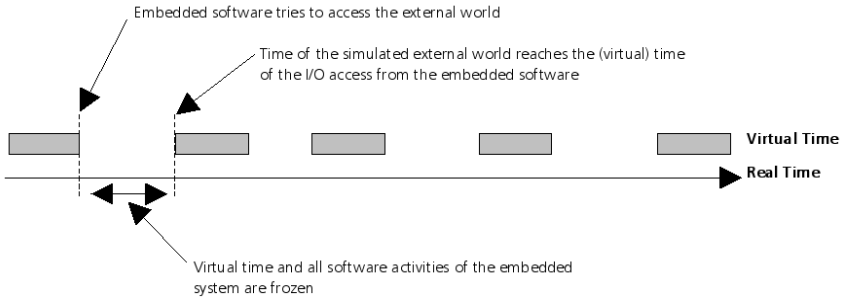

Figure 8: Two notions of time

To synchronize the virtual time and data in the embedded software with the simulation time and data in the simulator, we introduced a communication protocol which combines data transfer and time synchronizations, as shown in Figure 9.

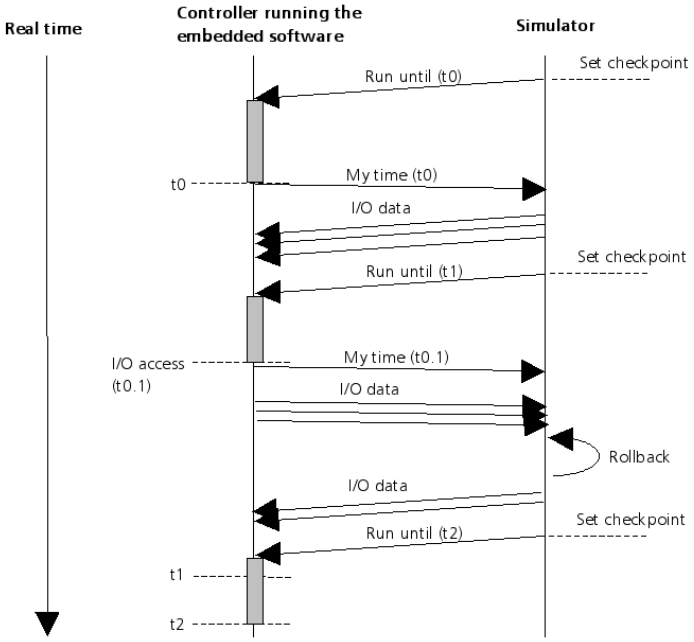

Figure 9: SiLEST synchronization protocol

The simulator begins the the simulation and continues until a time point (let us call it T0) at which it detects an asynchronous signal (data) going to the embedded controller. This might be, say, certain conditions generating an interrupt in the controller. The simulator thus knows when it will interrupt the controller, but it does not know when the controller will access data from I/O devices. The simulator allows the controller to run until a determined time point (message "Run Until(T0)" in Figure 9). The controller runs until T0 at the most, at which point the virtual time manager stops the time and all other software activities in the embedded controller. It reports the time to the simulator and waits for further instructions. At this point, the simulator transfers data to the controller. The simulator then gives the clearance to continue running until T1 (message "Run Until(T1)"). If, before this time point is reached, the embedded software attempts to access an I/O device (this is mostly the case), the virtual time manager again stops the virtual time and all other software activities, and then reports the time and the I/O access data to the simulator. The simulator receives the data and eventually it has to perform a rollback to the last checkpoint and resimulate until T0.1, which is before T1. At this point, a bidirectional data transfer is carried out to synchronize data and time. After these synchronizations, the simulator gives the clearance to continue until the next asynchronous signal – T2 in Figure 9 – which may still be the same as T1.

According to this protocol, there are two reasons for stopping the virtual time: I/O access and the reaching of the maximal simulation step given by the simulator.

5. Conclusions and Outlook

The SiL test approach presented here dispenses with the need for specialized hardware, making it suitable for deployment early on in the development process. Also, its use of synchronization between the controller and the simulation makes it possible to take a more precise look at the state of the software than is possible by an HiL test environment. Debugging potential in particular is enhanced by the ability to execute the software and simulation in steps. The chosen approach also makes it possible to obtain consistent states of the environment and the software under test. The well-defined layer structure of the simulation environment allows a library of simulation modules to be compiled, which can be used for HiL/SiL tests and, at the same time, is suitable for cross-project application. This enables costs to be cut during test preparation.

The SiL test's practical potential and limitations will only become evident towards the end of the SiLEST project, when the test process is tried out on software from the automotive and aerospace application domains. At present, it can safely be stated that real-life and HiL testing cannot be completely replaced by SiL testing because previous experience has shown that simulation and reality always differ. However, much of the environment simulation used in SiL testing will also be suitable for use in HiL tests and the flexibility offered by a totally software-implemented test environment will outweigh the drawback of the additional effort required by early error detection and the improved debugging options.

6. Recommended further literature and references

Briess K, Baerwald W, Gill E, Halle W, Kayal H, Montenbruck O, Montenegro S (2003) Technology demonstration by the bird-mission, ISBN 3-89685-569-7

Kumar K, Goswami, Ravishankar K (1993) Simulation of Software Behaviour Under Hardware Faults. Proc. of the 23rd Int. Symp. on Fault Tolerant Computing. 1993. pp. 218-227.

Poncet, JC (2000) Using Simulation to Design Real Time Applications. In Simulation in Industry'2000, 12th European Simulation Symposium 2000. Ed. D.P.F. Möller. Hamburg, Sep. 2000. pp. 43-47

Raguideau J, Schoen D, Henry J (1994) an Event-Driven Simulation Tool for Testing Software. In 5th Int. Symp. on Software Reliability Engineering. 1994. pp. 259-263.

SILEST Web presentation: www.silest.de

Zimbelman D, Anderson M, Correl T, Schnurr R, Fennel M (1996) The Attitude Control System Test-Bed for SWAS and future SMEX Missions. In Guidance and Control. 1996. V. 92. pp. 145-163