The Avionics System Test Bench, Functional Engineering Simulator: New Developments in Support of Mission and System Verification

11th Int. WS on Simulation & EGSE facilities for Space Programmes
SESP 2010

28-30 September
at ESTEC, Noordwijk, the Netherlands

João Rebelo[(1)], Quirien Wijnands[(1)], Francesco Pace[(2)], Robert Blommestijn[(1)]

[(1)]European Space Research and Technology Center (ESTEC) – European Space Agency (ESA)
TEC-SWG
Keplerlaan 1, 2200AG Noordwijk
The Netherlands
Email:Joao.Rebelo@esa.int

[(2)] GMV A.D.
Calle Isaac Newton 11, 28760 Tres Cantos, (Madrid)
Spain
Email: fpace@gmv.com

INTRODUCTION

The Avionics Systems Test Bench (ATB) is an ESA/ESTEC internal development of an avionics system test bench aiming to support the demonstration and validation of upcoming space avionics related standards and technologies in a representative environment, supporting projects in their need of assessing particular technology related issues and to provide hands-on training facility for ESA staff. Within the context of ATB, space avionics encompasses data handling (processing and storage), telemetry and telecommand processing, AOCS and mission management. The ATB is available in 4 configurations, namely: Functional Engineering Simulator (FES), Functional Validation Bench (FVB), Software Verification Facility (SVF) and Real-time Test Bench (RTB).

The Functional Engineering Simulator (FES) main purpose is the verification of the AOCS algorithms and the establishment of AOCS test scenarios for system tests at a later stage and, as such, it can be also a direct support to the OBSW development. The previous design was highly tailored for the Virtual Spacecraft Reference Facility (VSRF) earth observation reference mission, which resulted in a configuration which was difficult to modify, increase or adapt [1].

In the context of supporting missions like SGEO and the European Student Moon Orbiter (ESMO), functionality-wise and quality-wise improvements were needed. As the result of the Functional Engineering Simulator Enhancement (FESE) activity carried out by GMV and a follow-up activity by ESA, a generic and modular simulator infrastructure based on Matlab/Simulink® R2008b was created.

The main objectives for this development were to obtain a consistent architecture and interface definition as well as a centralized and easy-to-control database for keeping control over the parameters, interface definitions and signal logging. The intermediate results of the ongoing "Space Simulation Reference Architecture" (SSRA) activity were used as guidelines for the new design to ensure maximum consistency. Following the architectural improvements, an improved infrastructure is implemented, allowing for easy instantiation of a FES for a new mission, automated execution of simulation runs and results reporting. Also an interface to an open 3D visualization system was implemented to facilitate the visual analysis of the simulation results.

The paper describes in more detail the above mentioned enhancements, the trade-off analysis and the results of the enhancement activity and is organized as follows: Chapter 2 gives an overview of the simulator and space element equipment architectures and design decisions; Chapter 3 details the developments, enhancements and new features of the infrastructure; Chapter 4 describes the final results and future use for the developed infrastructure.

SIMULATOR ARCHITECTURE

In order for the FES architecture to cope with a number of different types of missions (in particular a geostationary, a navigation and a planetary exploration reference mission) with no or minor adjustments, a generic architecture was foreseen.

The developed architecture was divided, at top-level, in three main components, namely Ground Segment, Space Segment and Universe, as shown in Fig. 1. The Universe block contains all the calculations relative to time and celestial elements positions, velocities and other physical outputs and its values are available to the whole simulator via the global *Goto* block. The Ground segment represents the Earth ground and control stations, whereas the Space segment contains all the elements residing in space (e.g. spacecrafts, landers, rovers etc..).

The top-level space elements architecture was divided in environment, equations-of-motion and equipment, as illustrated in Fig. 2. This separation is expected to guarantee a strong decoupling between the environment subsystems, spacecraft motion and equipment, thus allowing for the easy replacement of one subsystem model without significantly changing the remaining simulator. The architecture of each space element is based on the idea of easy replacement between models with different levels of detail; therefore, all the subsystem models' inputs and outputs are represented by only three types of buses: 1) physical, 2) communication and 3) power. Type 1) contains all the physically related values, e.g. position, speed, torques; 2) is composed by all the value that are transmitted or received by using some type of communication bus, e.g. measured orientation, commanded speed, status data; 3) contains all the values related to the power bus, e.g. power consumed, power available or voltage. Using these abstractions all the interfaces can be represented and the model can be replaced with minimum effects on the overall simulator.

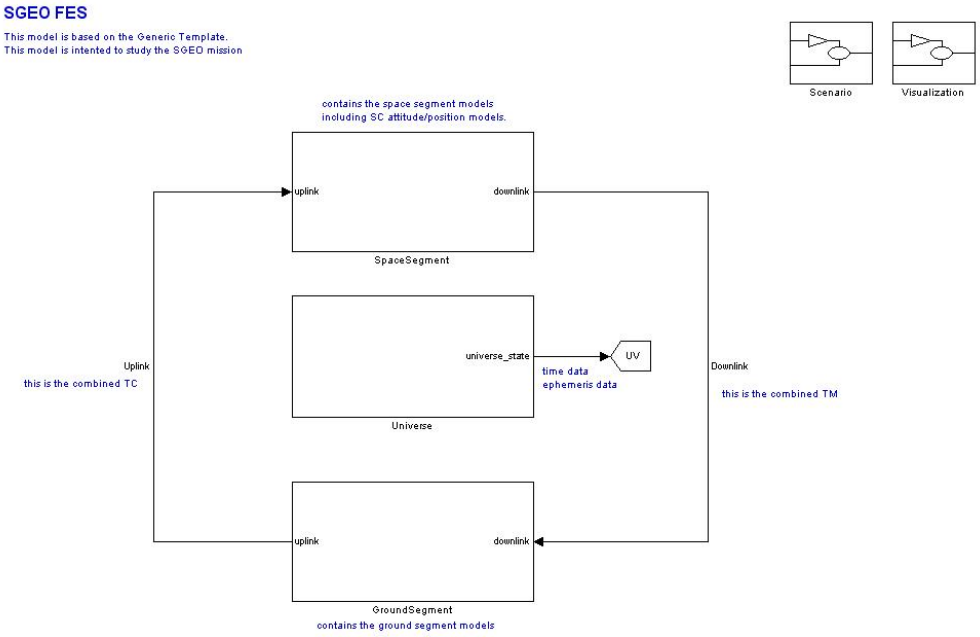

Fig. 1 Simulator top-level architecture

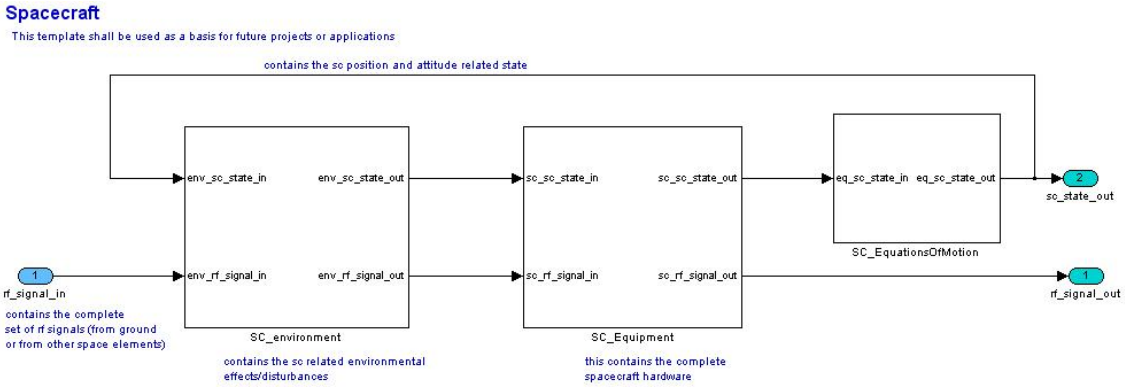

Fig. 2 Spacecraft architecture

Space Element Equipment

Given the proposed architecture and modularity level, a new set of dynamics, kinematics, environmental, sensors and actuators models was developed. Analyzing the intermediate results of the SSRA activity a generic equipment model, as depicted in Fig. 3, was defined. It can also be observed that not only the nominal behaviour of the elements is modelled, but also fault situations that closely mimic real hardware failures.

For implementing the models subsystem two possibilities exist: library blocks or model reference blocks subsystems [2]. Some of the main features of each of these types of blocks are shown in Table 1.

Even though model reference blocks have some advantages when building systems with many instances of the same elements, such as a spacecraft simulator, some problems were encountered. The main reason for these problems was the appearance of algebraic loops which the system could not solve. The analysis of potential causes and consequences of algebraic loops is out of the scope of this paper, and a more detailed explanation about it can be found in [3]. Nonetheless, the existence of algebraic loops led to unpredictable behaviours from the solver and inconsistent simulation results. For this reason all the subsystems were implemented using library blocks.

When creating the models all the constants are defined as variables, which are mapped to parameters in a mask. This parameterization permits easy behavioural adjustments, thus allowing an easy re-use of the subsystems' models both between different simulation runs and simulator instantiations.

To prevent errors due to wrong interface a strict interface control is implemented using Simulink Bus objects. To each input and output is attributed an object that guarantees that the simulation will only run when the signals on these ports have an identical structure to the one defined in the bus object. This is very important for all the models, in particular for the onboard computer, whose interface is very large and therefore hard to control if an automated method is not used.

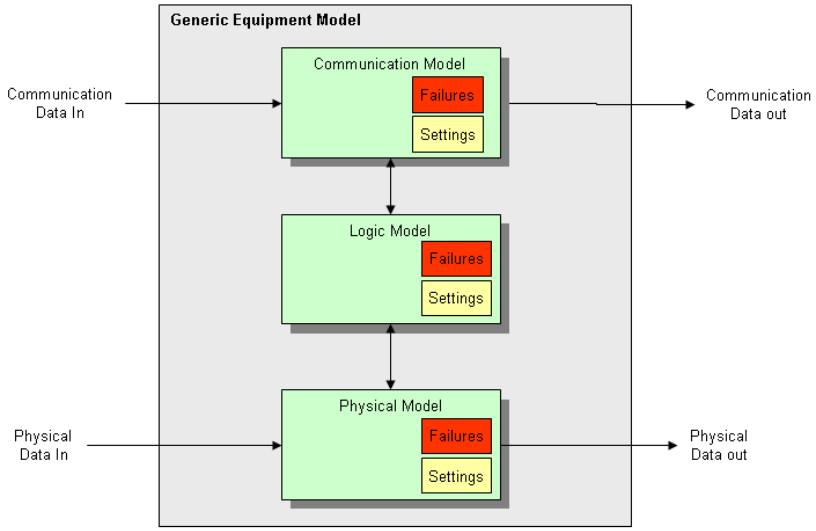

Fig. 3 Space element equipment architecture

Table 1 Library and model reference block features

Library blocks	Model reference blocks
Duplicated on every instance	Single instance
Executes together with the rest of the system	Executes as stand-alone unit
Similar to a macro in C	Similar to function in C

Depending on the type and use of the Functional Engineering Simulator, equipment and environment models might need different degrees of fidelity. To address this situation, the following fidelity levels are assumed:

- Simple: The simple model aims at being a continuous implementation that contains only the most essential functionality. The usage is mainly for AOCS/GNC engineers to design and develop the typical attitude and orbit control algorithms.
- Basic: This type of model contains first order features, including non-linear behaviours. This model will be used in early project phases, for example to support feasibility studies and preliminary design analysis.
- Advanced: These models provide the highest accuracy and similarity to the real spacecraft hardware. The objective of these models is to support detailed design analysis, onboard software development and verification and validation. These models should contain all the needed communication interfaces.

To guide the design and implementation of the simulation model, a coding standard was also defined. This should minimize the risk of compatibility and code-generation problems and at the same time lead to homogenous and maintainable simulation models.

FES INFRASTRUCTURE

Following the previously presented architecture and modelling guidelines, a FES infrastructure was developed to automate the database usage, failure injection, plotting and reporting mechanisms. This infrastructure is composed mainly by Matlab scripts, together with the use of an Excel spreadsheet used as database for defining parameters and bus objects, as well as controlling the simulation signal logging.

Database

To decouple the simulator functionality from the data and to guide the design of new models, all the models' parameters and interface definitions, as well as the logged signals, are kept in a database to provide a consistent and traceable baseline for each simulation campaign. The database also includes metadata, e.g. descriptions, units and reference frames, both to the signals and the parameters in the simulator. This database is currently implemented using an Excel spreadsheet from which *.m* files are created. Fig. 4 shows an example of a parameter definition (right side) using the database (left side).

Failure Injection

As stated in the previous chapter, for testing FDIR algorithms and performing some failure situation analysis each spacecraft subsystem was modelled to include failure conditions included (Fig. 3). The enabling of the failure conditions is done by using a Matlab script that can trigger the specific fault, either on time or on specific events in the simulator. The so-called *scenario script* is integrated in the top-level simulator architecture, as shown in Fig. 1, which permits the testing of nominal cases and different failure situations without making any changes to the overall model or parameters.

The disadvantage of this method is that the failures have to be modelled together with the subsystems and the inclusion of new failures can lead to major changes in the subsystem model.

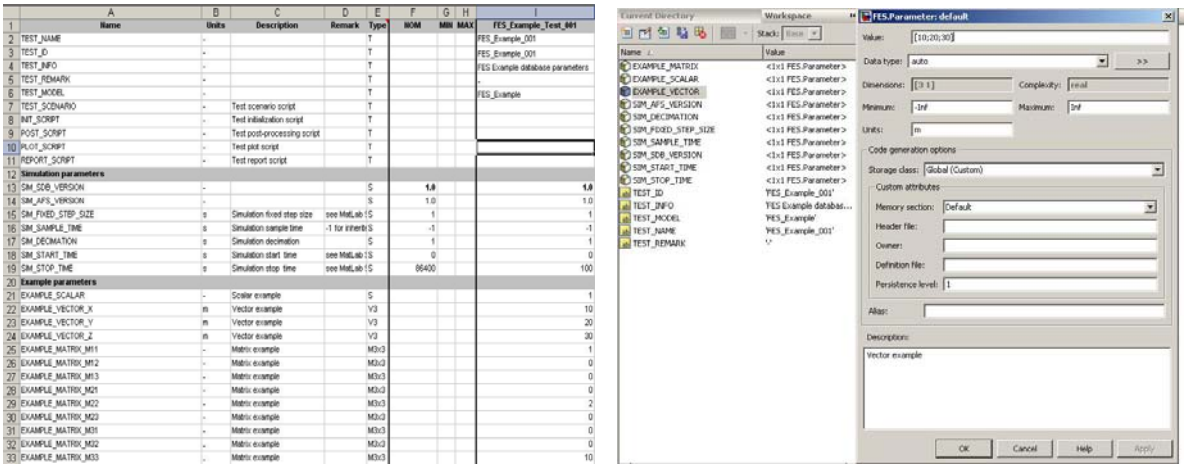

Fig. 4 Database parameter definition

Unit Test Framework

To facilitate the verification and validation of each of the individual subsystem and environment models a standard *unit test framework* was implemented. This framework stimulates the model under test with user-defined values, created by using Simulink Signal Builder. The actual outputs are then compared with the actual results and a pass/fail value is returned.

Fig. 5 and Fig. 6 show an example of the unit test framework and the usage of the signal builder block, respectively.

The use of the Signal Builder block, allows for easy change of inputs signals and creation of various signal groups which allows the execution of many different test without effectively changing the test harness. A model coverage test and a coding standard verification using Simulink Model Advisor were also included in the unit test framework.

Plotting and Reporting

The publication and sharing of results is also facilitated by including a plotting and auto-reporting facility. The plotting facility provides functions to create the most common plots in a spacecraft simulator and provides an open framework to be extended to define new plots, depending on users needs. In this way all the original Matlab plotting functionalities are still available. Fig. 7 shows an example of a plotted signal.

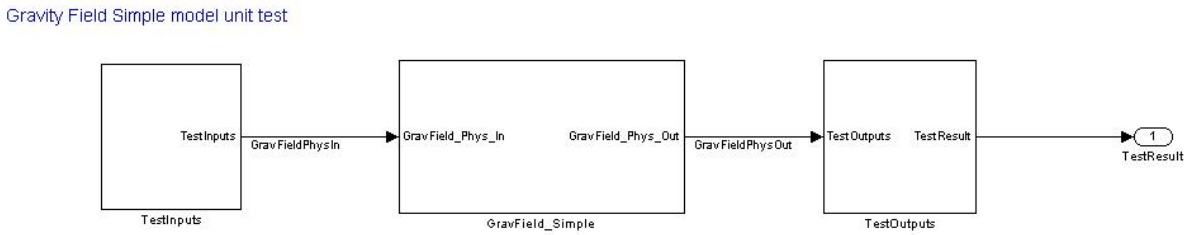

Fig. 5 Unit test example

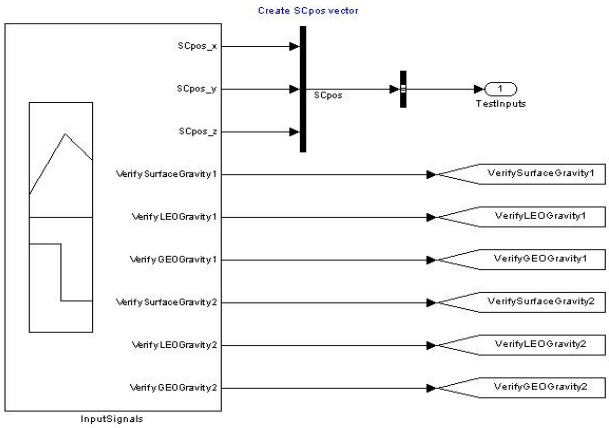

Fig. 6 Signal builder usage in unit test

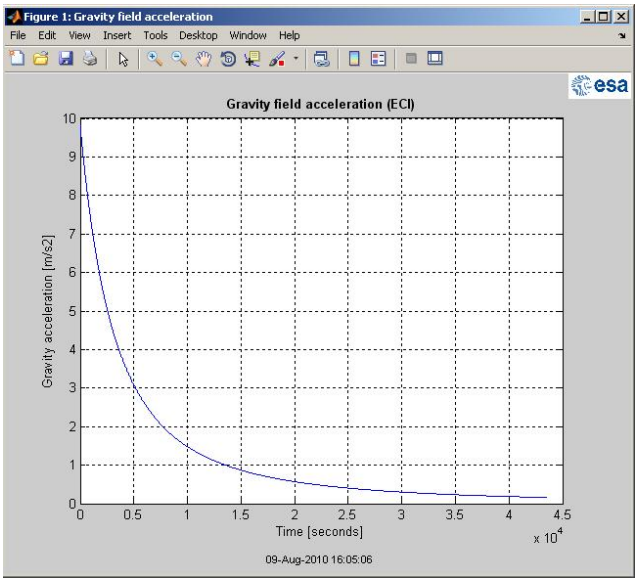

Fig. 7 Plot example

Note that the plotted figure includes information relative to units and reference frames, automatically generated from the database. This automated process keeps the consistency between the information in the database and the information that is plotted.

The result sharing functionality that automatically generates a simulation results report was implemented. This function currently updates a Microsoft Word document with the output figures and results, parameters, simulation date and version, configuration details and other simulation information. In this way, simulation reports are always kept consistent and up-to-date and results can be analyzed and shared without the need of re-running the whole simulation or re-installing the infrastructure.

Documentation

Another strong focus of the FES Enhancements activity was on the quality and accessibility of the documentation. For each of the implemented subsystem modules a set of system requirements, software requirements and detail designed documents were created. A direct link between model and requirements was created, allowing the requirements document to be easily accessed from the model and vice-versa.

The simulation facility provides a set of Matlab scripts which automatically build up the subsystem modules on-line help in HTML format according to a user defined template. This help can be browsed by the user and contains information of requirements, interface and detailed design for each library model extracted directly from the Word baseline documentation where bookmarks define the information to be reported in the help for each module.

The user manual is integrated in the Matlab help mechanism and each model's specific help document can be also accessed during design time using the integrated Matlab Help button. This way documentation can be easily consulted, thus easying the process of understanding the implemented features and selecting the appropriate models.

Visualization

To allow for visualization of the results using OpenIGS during simulation time, a visualization block in C was implemented. This block implements a buffer that collects the needed simulation data and sends it by TCP/IP to the visualization application. The buffer implementation is also responsible for sending the data for the correct time that has to be rendered. The overall architecture is illustrated in Fig. 8.

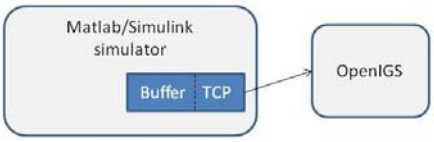

Fig. 8 Visualization system architecture

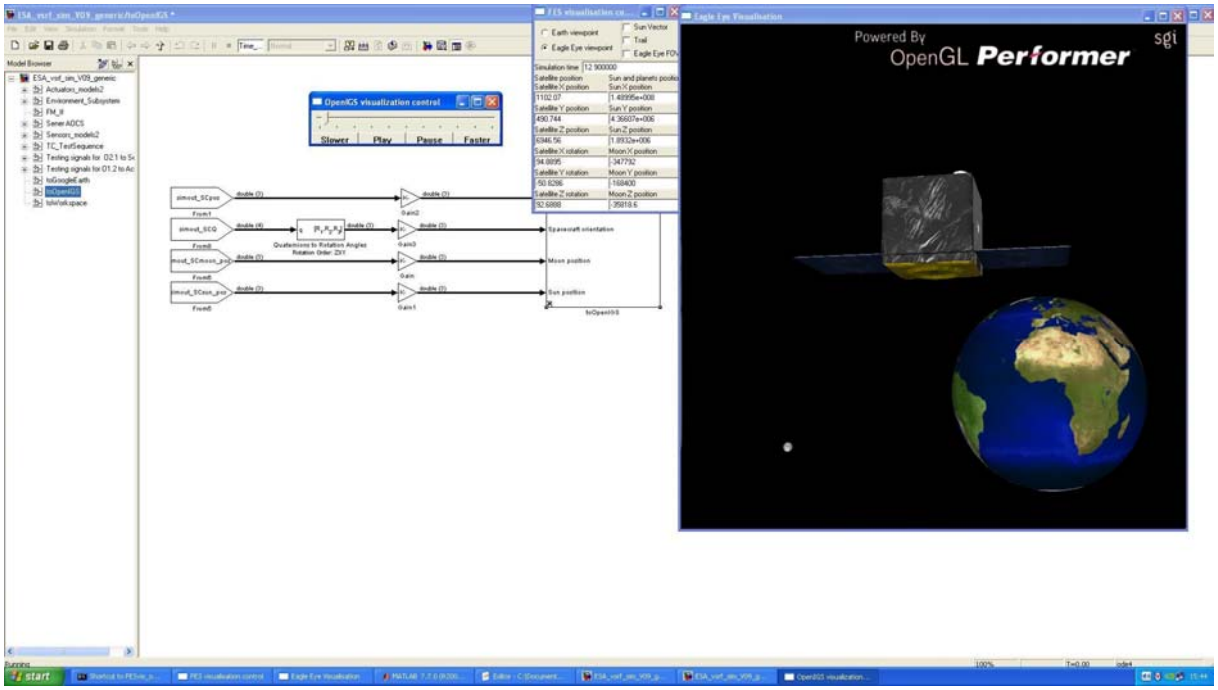

Fig. 9 Online visualization example

To allow for visualization of the results using OpenIGS during simulation time, a visualization block in C was implemented. This architecture has also the advantage of permitting any other visualization system that can receive the same type of messages to be used instead of OpenIGS. Fig. 9 shows an example of the online visualization system.

CONCLUSION

The results of both the FESE activity and the in-house improvements to the infrastructure have resulted in a highly modular environment for creation and execution of Functional Engineering Simulators. Although the modelling process has now some increased overhead mainly due to the interface definition and unit testing, the obtained models are much more reliable and re-usable. The proposed architecture is also proven to be easily adaptable, both when changing between different models' fidelity level and when testing different failure scenarios.

The new FES infrastructure is now considered stable enough for supporting the verification and validation of projects, even though some minor, project specific improvements are foreseen.

REFERENCES

[1] TEC-SWM, "ATB Functional Engineering Simulator Enhancements Statement of Work", 2008
[2] J. Friedman, *Power Tips For Using Simulink in a Large Project*, The Mathworks, 2008
[3] The Mathworks, "Matlab User Manual – Referencing a Model", 2008