The Lisa Pathfinder Simulator for the Science and Technology Operations Center: a Case-study for Reuse Across the Project Life-cycle

10th International Workshop on Simulation for European Space Programmes SESP 2008

7-9 October 2008 at ESTEC, Noordwijk, the Netherlands

Cristiano Leorato[(1)], Peter van der Plas[(1)]

[(1)]ESA/ESTEC
Keplerlaan, 1 - 2201 AZ Noordwijk, the Netherlands,
Email: Peter.van.der.Plas@esa.int

INTRODUCTION

During the operational phase of the Lisa Pathfinder (LPF) mission, the Science and Technology Operations Center (STOC) will be in charge of the operations of the LPF experiments. For the STOC to be able to perform its planning activities, an experiment simulator including the spacecraft and its environment is required. The STOC simulator will support the validation of the LPF Technology Package (LTP) run procedures and the data analysis activities. It will execute the run procedures generated by means of the MOIS tool ([11]) and provide measurements to the LPF Data Analysis tools. The STOC simulator will also be used for the training of STOC personnel and to incrementally update the modelling of the LPF spacecraft and in particular of its LTP experiment using flight data.

Also with the expectation to reduce the cost of the simulator development, it has been decided to base the experiment simulator on two other simulators that are already available in the LPF project. The Drag Free and Attitude Control System (DFACS) is a Windows Matlab/Simulink based simulator for the analysis of the DFACS performance and has served as a prototype for the onboard control algorithms. The Software Validation Facility (SVF) is part of the Astrium Model-based Development and Verification Environment (MDVE), where it is used for the integration and debugging of the onboard software in its stand-alone configuration.

REQUIREMENTS GATHERING

The reuse of the DFACS and SVF facilities is the main constraint in the development of the STOC simulator. The DFACS and SVF provide two alternative simulation strategies, which are highly complementary in terms of efficiency and accuracy of the results.

The main intent of the STOC simulator is to provide, in a unique framework, the seamless integration of the underlying simulation strategies, together with common simulation services (e.g. recording, monitoring, injection of TC sequences, injection of external stimuli, and definition of parameters of the simulation models). These simulation services will necessarily rely on different mechanisms on the DFACS and on the SVF, but it is the objective of the STOC simulator to hide as much as possible to the user the different implementation details provided by the two systems. The inputs to each service (e.g. variables or TMs to be recorded, initialization of parameters in the simulation models, etc.) are provided by the user defining a set of artefacts by means of a common user-friendly Man Machine Interface (MMI), as shown in Fig. 1. This allows the final user to more easily focus on his core activities.

It is expected that artefacts will evolve during the preparation of the LPF mission and even during the LPF mission itself. A typical example is the tuning of the parameters of the LTP models, which is expected to occur during the operational phase of the mission. Being LPF a demonstrator primarily aimed at LISA, the physics models will be a significant part of the LPF legacy for the LISA mission. The configuration control of the simulation and of the associated artefacts shall therefore be part of the STOC Simulator.

In general, several users might be willing to utilize the STOC Simulator at the same time, but the system shall also be portable, allowing the user to check out a set of artefacts to a stand-alone PC, and to run simulations without the continuous support of the configuration control system. In this case, some configuration control issues may have to be manually handled at the end of the process (e.g. merging of concurrent modifications on the same artefacts, etc.).

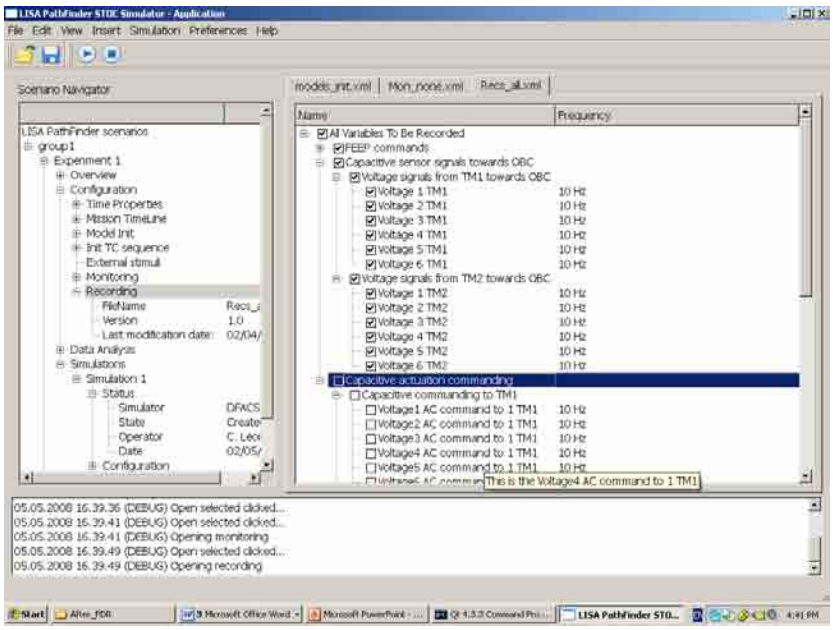

Fig.1. Snapshot of the STOC Simulator MMI prototype: editing a recording artefact

The detailed requirements of the system were gathered on the basis of several informal interviews with the different stakeholders (see [2]). In order to cross-check with the users the understanding of the required system, a more structured meeting was organized before the Preliminary Design Review meeting. On this occasion, a prototype of the product was presented, allowing discussing more concretely the functional aspects involving the MMI. As shown by the snapshot reported in Fig.2, the prototype also helped the detailed discussion of complex use cases, involving running simulations and handling the configuration control of the associated artefacts.

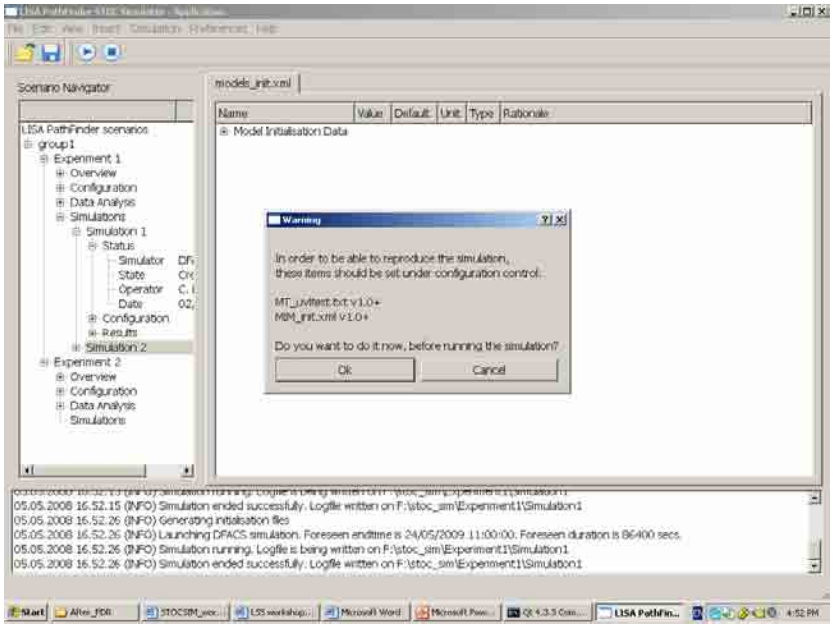

Fig.2. Snapshot of the STOC Simulator MMI prototype: configuration management of simulations and artefacts

RUN-TIME TECHNOLOGIES AND REUSE APPROACH

The reuse of the DFACS and SVF simulators is a direct consequence of the user requirements.

The DFACS simulator runs on Windows and relies on the Matlab/Simulink products. It is a design simulator, originally conceived for the analysis of the Drag Free and Attitude Control System performance, and served as a prototype for the on-board control algorithms.

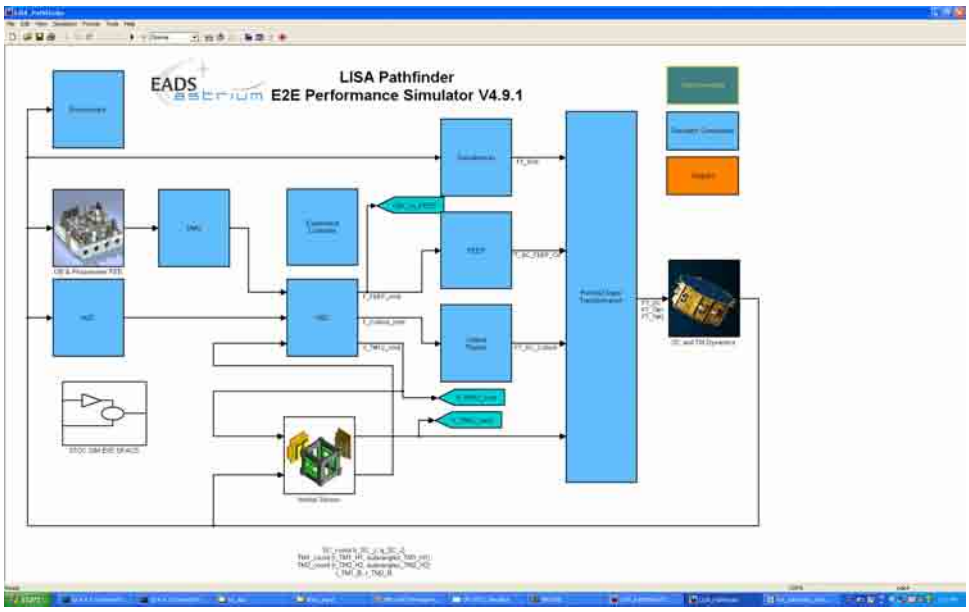

Fig.3. Snapshot of the Simulink-based DFACS simulator

It shall be noted that, although the current DFACS simulator status is rather stable, modifications from the developers can still be expected. In order to limit the impact of configuration control issues, the DFACS extensions needed for the STOC purposes (e.g. monitoring, recording, etc.), whenever possible, do not depend on the DFACS simulator and are directly based on the Matlab/Simulink layer.

Given the more limited scope of the DFACS simulator w.r.t. the operational context, an adapter layer has been implemented, in order to be able to map, when needed, the MOIS TCs to their counterparts in the design simulator.

The SVF facility runs on Linux and actually consists of two products: the SVF Real Time Simulator (RTS) and the Open Center application. The SVF RTS comprises the simulation models and the real OBSW, running on an emulated OBC.

Open Center is a generic EGSE, which, in the context of the Lisa Pathfinder SVF facility, is the main means to interface the SVF RTS: sending TCs, acquiring TMs, and, more in general, perform any simulation monitoring and control. The main Open Center Front End is based on a complex MMI. This makes it difficult to hide the overall complexity of the SVF facility to the STOC user. It is actually expected that the STOC team will have to perform a certain number of tasks by directly operating the Open Center environment. Nevertheless, a language script, Elisa, is also available and can be used to automate a certain number of tasks. For example, the TC commanding, according to a given run procedure, can be implemented by means of an Elisa script.

Apart from those inherited from the DFACS and SVF facilities, a certain number of other products and technologies were chosen by the development team in the context of the project. In particular, the reuse of the DFACS and SVF simulators determined the need to prefer cross-platform technologies, addressing, as a minimum, the target platforms of these simulators.

C++/Qt (see [8]) was chosen to provide the core of the STOC Simulator. Qt is a cross-platform library mainly aimed at MMI development, but also providing other interesting functionalities, such as an implementation of containers and generic algorithms, of platform-independent Inter Process Communication mechanisms, and an increasing support to XML standards. The C++/Qt framework is well documented and has a wide user community. It satisfies the cross-platform requirements without the need of a specific Virtual Machine, enabling high performances and a very easy installation and deployment of the final product.

CVS is the chosen product to implement the configuration control of the artefacts managed by the users during the preparation and validation of the run procedures. The main reason for this choice is that CVS is already the preferred configuration control tool used by the STOC. Furthermore, CVS is a cross-platform and mature Open Source product, and promotes a concurrent development model (not necessarily based on “reserved checkouts”, see [7]), which should enable a more efficient development of the simulation artefacts by the overall STOC team.

The chosen framework for data representation is XML. Once again, this choice was driven by the preference for a platform-independent and light-weighted framework. Moreover, part of the initialisation data in the SVF facility is also provided in XML. As we have already mentioned, the Qt library also provides support for the XML technologies. After

considering the usage of alternative XML-specific libraries, it was decided to strive for simplicity, maximising the cohesion of the run-time environment: dependencies on additional libraries may still be added in the future, if special needs arise (e.g. need of a higher performance XQuery framework).

Fig. 4 summarises the run-time technologies integrated into the STOC simulator. It also clearly identifies those parts of the system which are developed for the STOC simulator project:

- The STOC Simulator library is by far the most complex component, as it provides most of the functionality required to implement both the common STOC Simulator MMI and the associated algorithms.
- The STOC Simulator MMI relies on the STOC Simulator library and on the Qt environment.
- The DFACS STOC Simulator Executor provides the DFACS extensions required for the STOC users.
- The SVF STOC Simulator Executor provides the SVF extensions required for the STOC users.

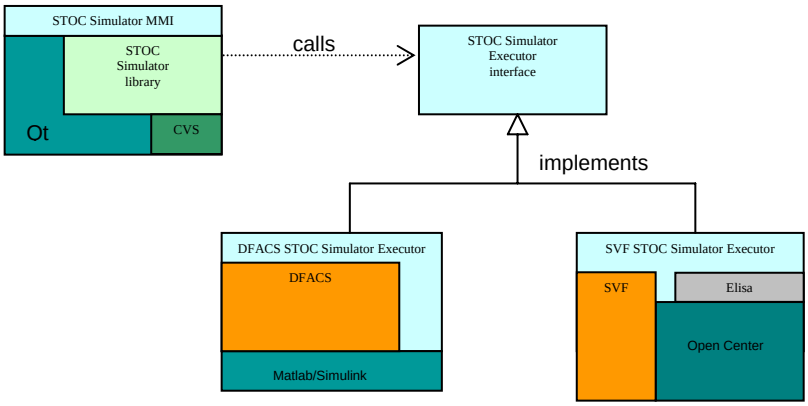

Fig.4. STOC Simulator run-time technologies

THE DEVELOPMENT PROCESS AND TECHNOLOGIES

The development process as a whole follows the ECSS-E-40 standard as defined in [1]. In order to enhance the customer collaboration and to encourage its early familiarization with the system, several informal incremental deliveries of the STOC Simulator are also foreseen. According to Agile principles (refer to [5], [3], or to the case-study in [10]) this is expected to provide an earlier feedback from the users, facilitating further refinements of the system, and avoiding a dangerous Big-Bang integration. The first version of the STOC Simulator, including most of the common parts of the STOC Simulator and of the DFACS interface, was delivered in September 2008 and two more major deliveries are scheduled this year.

At the beginning of each development iteration, the latest Qt version is integrated into the up-to-date STOC simulator, allowing to get the most from the Qt technologies, with very limited regression risks on the product under development.

Apart from the products and technologies described in the previous section, a certain number of tools and products are specific to the development framework of the STOC simulator. On the Windows platform, the development is carried out on two different environments, namely Visual Studio (including the Microsoft Visual C++ compiler) and Eclipse CDT (using the MinGW compilation chain). The usage of two different compilers is a good and common practice to increase the reliability of the developed software.

Build and deployment are based on the platform-independent *qmake* tool provided by the Qt framework.

Striving for agility and according to the approach described for example in [4], UML, as a tool for upfront design, is mainly used to provide simple sketches of the system design. *Doxygen* and *graphwiz* (see [12] and [13]) are then used to provide automated documentation, which includes more extensive and detailed UML diagrams.

Whenever suitable, in order to avoid unjustified complexity, the same tools used for the run-time environment were also chosen as part of the development environment: CVS is the configuration control system used to manage the STOC simulator source code. Concerning XML technologies, together with Qt, the dedicated and very efficient XML library from Saxonica ([14]), also supporting XQuery 1.0 and XSLT 2.0 standards, is currently used to support the development, tuning, and validation of XML transformations.

As in all incremental processes, particular emphasis is put on the definition of the unit test environment. Given the issues implied by MMI validation, in this project, a pragmatic and light-weighted approach is adopted: almost all of the STOC Simulator development is based on the development of the STOC Simulator library, for which a complete set of automated unit-tests is designed and maintained. For this purpose, a suitable unit-test framework has been developed, extending the *QTestLib* library provided as part of the Qt framework. Finally, in order to validate the DFACS

extensions needed for the STOC purposes, an additional unit-test framework has been developed, based on the experience of the authors in the reuse and validation of Simulink S-functions (see [9]).

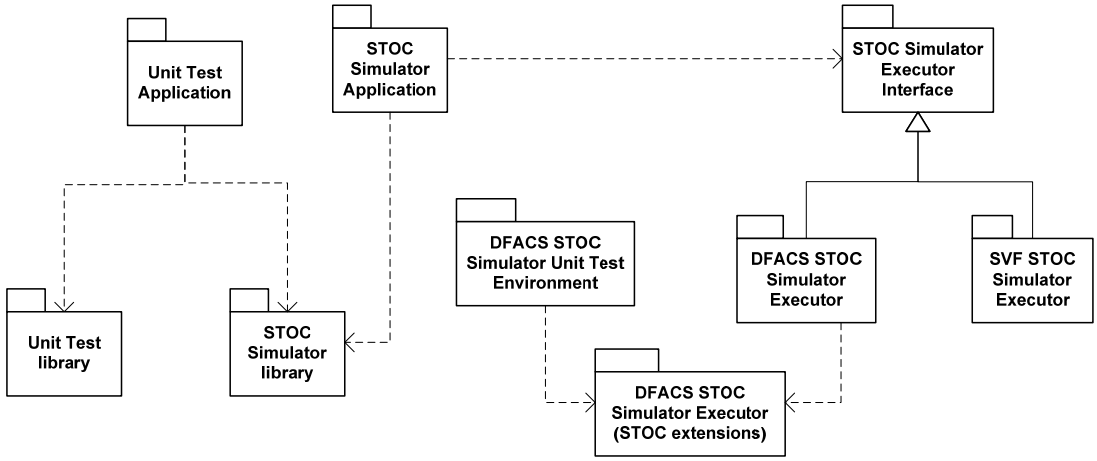

Fig.5. The unit-test approach

DESIGN APPROACH

Keeping agility principles in mind, a moderate number of design guidelines and patterns have been identified already before the Preliminary Design Review meeting. The analysis of the requirements has shown that commonalities exist, between the software components in charge of handling the different artefacts (e.g. TC sequences, model initialisation data, monitoring, recording). Most of the artefacts shall provide:

- A Data Model of the artefact (e.g. in order to load and save the artefact)
- A Data Editor MMI, allowing the user to view and/or edit the associated item in a user-friendly way.
- Specific DFACS and SVF functionality, in order to generate the data corresponding to the item, in the format expected by the underlying simulator.

One can benefit from the encapsulation of this interface, introducing an “Abstract Artefact” interface, as shown in Fig. 6. This provides a normalised interface between the STOC Simulator Kernel and the different types of artefacts. The STOC Simulator Kernel provides the generic framework for the STOC Simulator MMI, allowing the user to define and edit the simulations and the associated artefacts, but all artefact-specific editors and algorithms are implemented in the appropriate artefact objects. The STOC simulator Kernel also invokes the appropriate underlying simulation (DFACS or SVF).

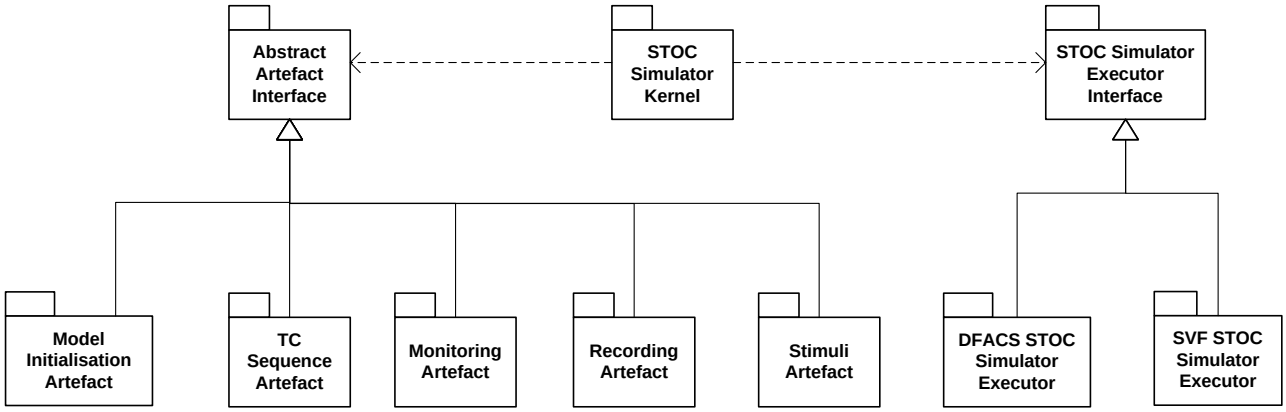

Fig.6. Main interfaces in the STOC Simulator design

Each concrete Artefact delegates the generation of specific DFACS and SVF data to appropriate components. As the user can modify the choice of the target platform anytime before running the simulation scenario, for flexibility reasons, the design follows the Strategy pattern defined in [6].

Concrete instantiations of Abstract Artefact are required by the STOC Simulator Kernel in two different circumstances:

- When the user requires editing a certain artefact. In this case, both a Data Model and the corresponding Data Editor shall be created. The creation shall be consistent with the type of artefact which needs editing.
- When the user requires the execution of a specific simulation. In this case, the STOC Simulator Kernel has to create the Data Model of all artefacts contributing to the definition of the configuration of the simulation. The creation of the artefacts shall be consistent with the underlying simulation (DFACS or SVF).

To ensure consistency, the Abstract Factory pattern (described in [6]) is helpful to create the appropriate set of components. For example, in the context of a simulation execution request, the creation of the associated simulation artefacts follows the design shown in Fig. 7.

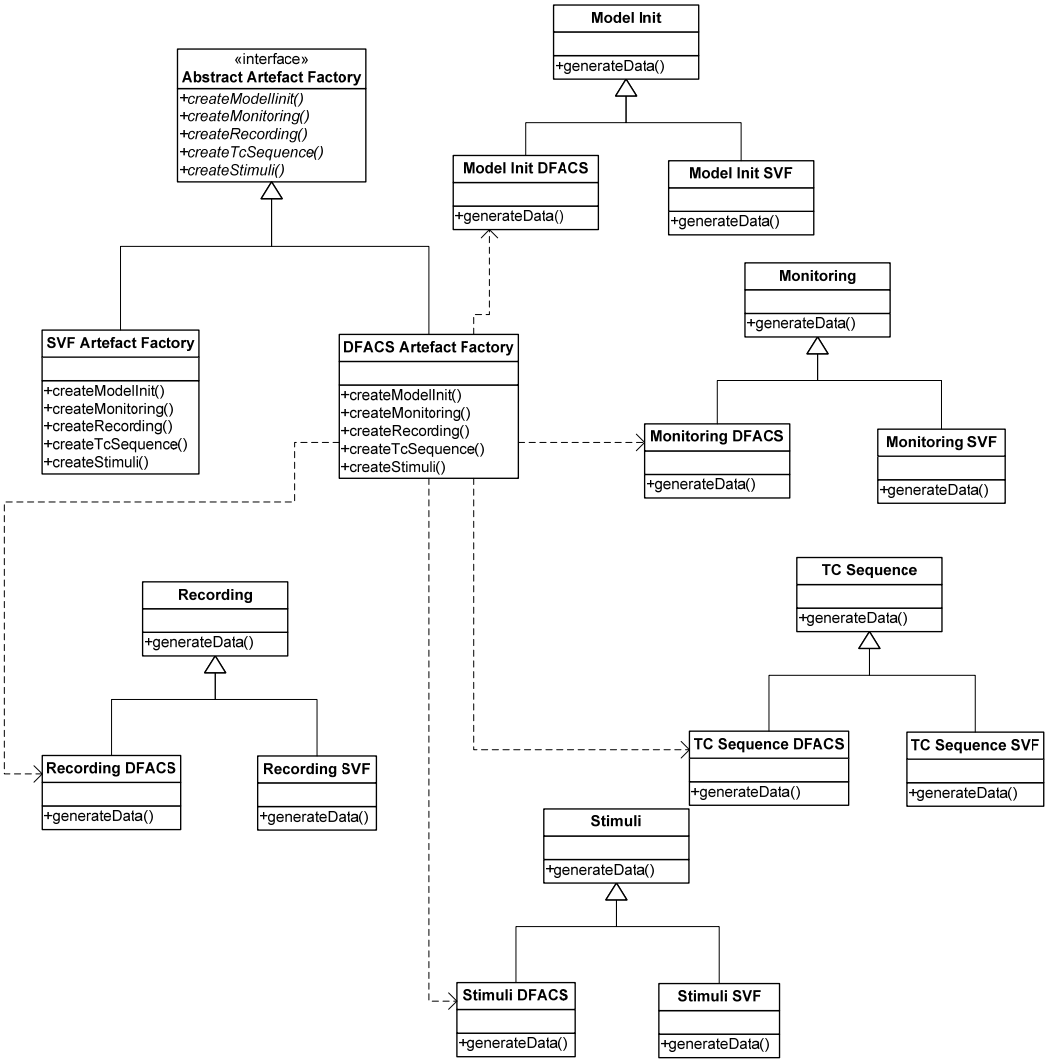

Fig.7. The Abstract Factory pattern to create consistent artefacts

CONCLUSIONS

The STOC simulator aims at developing a user-friendly simulation environment for the LPF science operations, by reusing the DFACS and SVF simulators, originally developed by the industry to evaluate the system performance and to validate the on-board software. Adequate solutions have been found for all reuse issues identified so far. This paper has illustrated some of these solutions.

The reuse of the DFACS and of the SVF is expected to significantly reduce the development cost of the STOC Simulator, and also has the benefit of increasing the coherency between the different simulation facilities used in the overall LPF project.

Moreover, this approach minimises many of the potential pitfalls and risks, which would have been introduced by the development of a completely new simulator: uncertainty on the level of fidelity provided by the new simulator, heavy cross-validation w.r.t. validated reference platforms, etc.

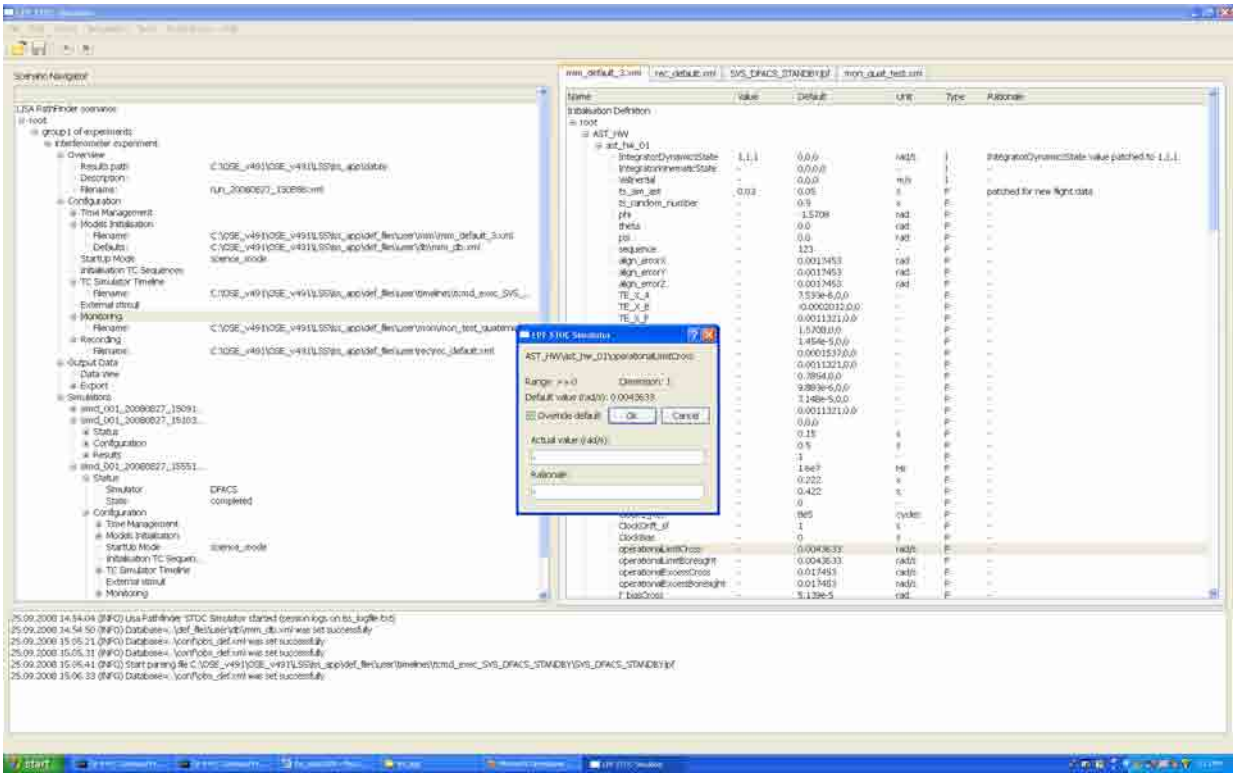

Fig.8. Snapshot of the STOC Simulator MMI: the editor for model initialisation data artefacts

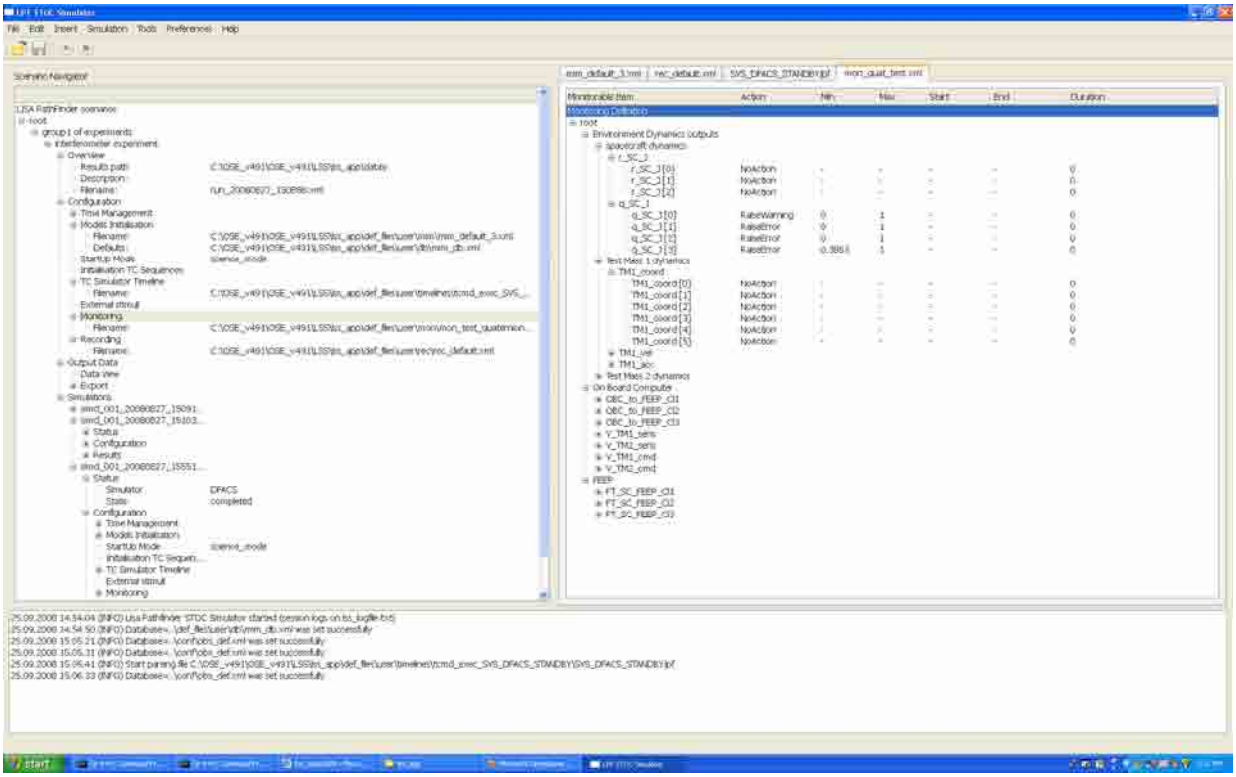

Fig.9. Snapshot of the STOC Simulator MMI: the editor for monitoring artefacts

REFERENCES

[1] "Space Engineering Software – Part 1: Principles and Requirements"-ECSS-E-40 Part1B.
[2] S. Robertson, J. Robertson, "Mastering the requirements process" 2nd ed., Addison-Wesley, 2006.
[3] M. Fowler, "Refactoring: Improving the Design of Existing Code", 3rd ed., Addison-Wesley, 2003.
[4] M. Fowler, "UML Distilled", 3rd ed., Addison-Wesley, 2004.
[5] R. Martin, "Agile Software Development Principles, Patterns, and Practices", 1st ed., Prentice Hall, 2003.

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns Elements of Reusable Object Oriented Software”, 1st ed., Addison-Wesley, 1995.
[7] J. Vesperman, “Essential CVS” 2nd ed., O’Reilly, 2006.
[8] J. Blanchette, M. Summerfield, “C++ GUI programming with Qt4”, 1st ed. Prentice Hall, 2006.
[9] C. Leorato, E. Guidolotti, D. Segneri, “Reusing Legacy Code in a Simulink Environment for Real-time Industrial Simulations: Techniques and Prototype Support Tools”, Proceedings of the DASIA Conference 2005, ESA SP-602 (August 2005) pp. 316, 327.
[10] C. Leorato, A. Huet, B. Sarter, “Design Patterns for Hardware In the Loop Simulators: the Common Model Driver and the IVF-EPOS Case-Study”, Proceedings of the 9th Workshop on Simulation for European Space Programmes, Noordwijk, 2006.
[11] O. Camino, R. Blake, W. Heinen, “Smart 1 Scheduler – A Cost Effective Mission Scheduler compatible with SCOS2000”, Proceedings of the 4th International Workshop on Planning and Scheduling for Space, ESOC, Darmstadt, 2004.
[12] “Source Code Documentation Generator Tool”; http://www.stack.nl/~dimitri/doxygen [Accessed 6th October 2008]
[13] “Graph Visualization”; http://www.graphviz.org [Accessed 6th October 2008]
[14] “About Saxon”; http://www.saxonica.com/documentation/index/intro.html [Accessed 6th October 2008]