# An Hybrid Design Solution For Spacecraft Simulators

Vítor Rodrigues[1,2], João Correia Lopes[3,4], Ana Moreira[5]

[1] ESOC/ESA, D-64293 Darmstadt, Germany
victor.rodrigues@esa.int
[2] Oristeba — Space Services
http://www.oristeba.com
[3] INESC Porto, 4200-465 Porto, Portugal
[4] Faculdade de Engenharia da Universidade do Porto, 4200-465 Porto, Portugal
jlopes@fe.up.pt
[5] Dept. Informática, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
anm@di.fct.unl.pt

**Abstract.** The European Space Agency (ESA) has created the Simulation Model Portability 2 (*SMP2*) standard with the purpose to provide a design solution for the project of Spacecraft Simulators. One element of the *SMP2* standard is the metamodel Simulation Model Definition Language (*SMDL*). The design artefacts of a Spacecraft Simulator consist in descriptions of the business logic shared by a set of *SMP2* models. This paper reports results from a study that considers the hypothesis to complement the *model-driven* design approach of the *SMP2* standard with *test-driven* design techniques. The high-level abstractions of Spacecraft Simulators are used to carry out *Model-Driven Development* processes, while reusable pieces of software that can to be used by many *SMP2* models are designed and developed following *Test-Driven-Development*. The tool capable to establish the dependencies between the source code produced by the two methodologies and mission specific source code is the *GNU Build System*.

Keywords: simulation model portability, model-driven development, test-driven development, GNU build system, hybrid design techniques

## 1 Introduction

The design of Spacecraft Simulators is based upon a component model specified by the the *SMDL* modeling language which focuses primarily on interface reuse [1]. *SMP2* models are described in *SMP2* design artefacts that are transformed into C++ *skeletons* into which the behaviour implementations must be *added*. Although model-driven design is ideal for developing software in multiple computing platforms, multiple implementations of the same interface is left out of its scope [2]. Our objective is to *deconstruct* the purely top-down strategy of model-driven development using the *SMP2* standard onto a bottom-up development process of a *SMP2 Framework* [3]. On the other hand, reusable behaviour

implementations of a Spacecraft Simulator are developed on top of a distinct software framework called *Infrastructure Framework* developed with test-driven techniques. The hybrid design solution aggregates the principles of both software development approaches.

## 2 Background

Design prototyping in object orientation is the activity of using "objects" that represent abstract entities to define design models. From this premise follows that the planning of the coding activity is done by *describing* the objects and the business logic they share. Although the design models are written before any programming language specification, we can establish a direct correspondence between the symbology of a modeling language and the symbology of a programming language, thus making the design models a cross-platform specification from which model-driven development departures.

A different approach to software design is the specification of code functionalities through test code. Using only test code it is possible to design a piece of software before entering the stage of source code development. Similarly to design models, test code is a design artefact, but it does not follow a pre-defined semantic scheme as per design models. Nonetheless, test-driven development is a restrained process, carried out in closed loop through source code *refactoring* which makes it appropriated for developing software systems that are continually evolving.

This *GNU Build System* provides modelling languages to specify dependencies between *source packages* [4]. *Cross-platform* processes of creation of makefiles are integrated within the development of *SMP2* components, making possible the attempt of several configurations involving the *SMP2* models and the reusable libraries, which derive from disparate development lines.

## 3 Specification

Our premise is that there are parts of the spacecraft functionality that do not depend on the business logic configuration, because they are generic, context independent and, therefore, reusable. Therefore, the implementation of a Spacecraft Simulator, whether it is accomplished from scratch or as an update to an existing solution, may be decoupled from the design models.

An hybrid system would be possible to reconfigure by simply adjusting the variable parts that in it coexist. The articulation of both frameworks, that is, the integration of the reusable software *inside* the *SMP2* models, is accomplished by the *GNU Build System* (see Figure 1). The creation of a *SMP2* model is done indirectly. Template methods of the *SMP2 Framework* become hook methods on the specialized classes and the mechanism of object inheritance is used to plug-in software components into the framework's *hot spots* [5].

To build a functional *SMP2* software component it is necessary to link the code *skeletons* and the infrastructure libraries with "glue" code designated by

*mission specific*, which must reach the reusable behaviour provided by the infrastructure libraries and make the necessary adaptations to, in its own behalf, provide to the other *SMP2* components the behaviour which is specified in the *SMP2* logic structure. This activity constitutes the tuning of the system and the *GNU Build System* guarantees that the system is kept in a consistent state [6].

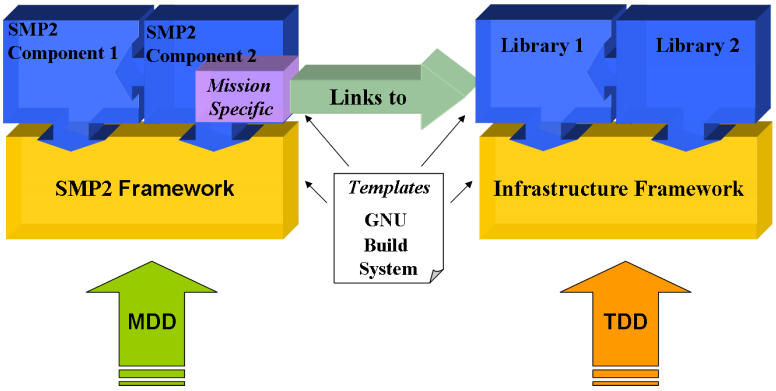


**Fig. 1.** The Development Lines of MDD and TDD

Two phases of decoupling are foreseen: the first decouples interface descriptions from code *skeletons*, and the second decouples code *skeletons* from behaviour implementations. The transformation between design models and code *skeletons* is automated by the model-driven development environment, but the linking to the behaviour implementations is not offered by the *SMP2* standard.

The access to the infrastructure functionalities can be automated by adopting the 'Adapter' and 'Abstract Factory' *GOF* [7] design patterns, reducing the amount of manual writing and software analysis: if the translation between the public *SMP2* interfaces and the private interfaces of the infrastructure libraries can be specified using the *SMP2* support for metadata, then this translation is foreseen as yet another transformation inside the *MDD* environment.

The hybrid system is developed by an iterative process. In each iteration it is possible to refine the *SMP2* interface signatures or the behaviour provided by the infrastructure libraries. The granularity of the *SMP2* models is decided upon metrics analysis and the granularity of the *Infrastructure Framework* is determined by the number of different contexts where a given library can be reused. If these two operations converge and if relation between the *SMP2* models and the combination of infrastructure libraries is of the type *adapter:adaptee*, then the additional coding of mission specific code will increasingly tend to zero [6].

With a pure model-driven design approach the initial prototype *becomes* the simulator after completing the coding task. This enforces the validation of the design in the earlier stages of the project life cycle [8]. On the contrary, the risk

of changing the business logic of an hybrid system during development is reduced since the core tasks of coding are done on top of the *Infrastructure Framework*. In such a decoupled system, the *SMP2* code *skeletons* can be regarded as *test* code, but there has to be no commitment to an initial high-level design.

## 4 Conclusion

A single design approach is hardly a one-size-fits-all solution. An hybrid system is more pluralistic because it provides the software engineers with a large set of "building blocks", which are designed to be useful in different application contexts and used to build software systems without a fully pre-fabricated structure. In this line of thought, the design of Spacecraft Simulators supported in the two distinct development methodologies of *MDD* and *TDD* widens the covering of the software requirements and produces a more complete project specification. The advantage of an hybrid solution is the opportunity to circumscribe the technological *push* of the *SMP2* standard and work exclusively on the design models and evaluate the impact that the modelling breakdown imposes. The *GNU Build System* is the enabling technology for an hybrid design solution, bringing flexibility to the integration of source code derived from the *MDD* and *TDD* development lines.

## References

1. ESA: Smp 2.0 Handbook. Technical report, EGOS-SIM-GEN-TN-0099 (October 2004) Issue 1 Revision 0.
2. Brown, A.: An introduction to Model Driven Architecture, Part I (2004) Online resource: `http://www-128.ibm.com/developerworks/rational/library/3100.html`, last accessed on 10-March-2007.
3. Johnson, R.E.: Components, Frameworks, Patterns. In: ACM SIGSOFT Symposium on Software Reusability. (1997) 10–17
4. Vaughan, G.V., Elliston, B., Tromey, T., Taylor, I.L.: GNU Autoconf, Automake, and Libtool. Sams; 1st edition (2000)
5. Pree, W.: Meta Patterns — A Means for Capturing the Essentials of Reusable Object-Oriented Design. Lecture Notes in Computer Science **821** (1994) 150+
6. Rodrigues, V.: On the Specification of Spacecraft Simulators using Object-Oriented Methodologies. Master's thesis, University of Oporto, Departament of Electrical and Computer Engineering (2007)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (2005)
8. Ambler, S.W.: The Object Primer. Cambridge University Press (2004)