A FIRST EXPERIENCE ON USING SMP2 FOR THE VIRTUAL SPACE SYSTEM FOR GROUND DATA SYSTEM VALIDATION

C. Laroque[1]
VEGA, Darmstadt, Germany

M. Pecchioli[2] **and N. Sebastiao**[3]
European Space Agency, European Space Operations Centre, Darmstadt, Germany

The Ground Systems Test and Validation Infrastructure (GSTVi) aims at a harmonized approach to development, configuration, operation and maintenance of test tools for ground systems in order to reduce the cost of ground segment validation. GSTVi consists of a set of simulation models which can be customised for specific missions and allow simulation of the complete telecommand and telemetry chain from the Mission Control System (MCS) to the spacecraft and vice versa. The set of simulation models implementing this "virtual space system" includes an MCS model, a Network Interface System (NIS) model, models simulating ground station TM/TC processing equipment, and a highly configurable model of a Packet Utilisation Standard (PUS) based spacecraft. Depending on the test configuration to be supported, one or more real ground systems can be introduced into the chain while the others are represented by their simulation model.

The objective of this concept is to support the full validation cycle of ground data systems, starting from the validation of each ground system in isolation, through incremental ground segment integration testing up to system level campaigns such as Mission Readiness, Operational Simulations Campaign and even routine pre-pass data flow tests.

Following a summary of the GSTVi concept, this paper presents the experience on using the new SMP2 standard for the development of the GSTVi system. It describes how the SMP2 standard has been used, how the GSTVi system has been designed to be SMP2 compliant, the benefits of using SMP2 but also the constraint it imposes on the development process. The experience on using the tools linked to the SMP2 development that have been used for the GSTVi development is also presented.

THE VIRTUAL SPACE SYSTEM CONCEPT

The traditional approach to the validation of ground systems relies on the use of various test tools that are developed, maintained and configured independently. Often, such tools emerge from ad-hoc developments responding to an immediate need in a project, are limited in scope, and require special knowledge for their operation. In effect, testers have to familiarize themselves with a variety of different tools, tests have to be designed in different ways to meet the capabilities of these tools, and test configuration data have to be redeveloped for each of these tools.

The Virtual Space System concept was motivated by the desire to exploit the synergy between different test tools and simulation models, to provide harmonized behaviour, a uniform, integrated user interface, and a single configuration database for all tools. The objective of this initiative is to

- Provide a coherent and more complete end-to-end test environment earlier in the lifecycle of the ground systems;

[1] Communication & Test Systems Group Leader, Europaplatz 5, 64293 Darmstadt, Germany
[2] Head of the Operation Centre System Infrastructure Section, Ground Systems Engineering, Robert-Bosch-Str. 5, 64293 Darmstadt, Germany
[3] Software Engineer, Ground Systems Engineering, Robert-Bosch-Str. 5, 64293 Darmstadt, Germany

- Provide the ability to validate each individual ground system in complete isolation from any other element of the ground segment;
- Support ground systems validation at the different stages of integration, including system testing;
- Maximize the re-use of configuration data across different tools and test configurations;
- Maximize reuse of infrastructure software across the different test tools supporting TM/TC data exchange;
- Harmonize configuration and operation approaches across the different tools in order to minimize the effort required for familiarization, training, test specification, and configuration.

It is expected that these improvements will help to reduce development and testing costs and increase the quality of ground systems.

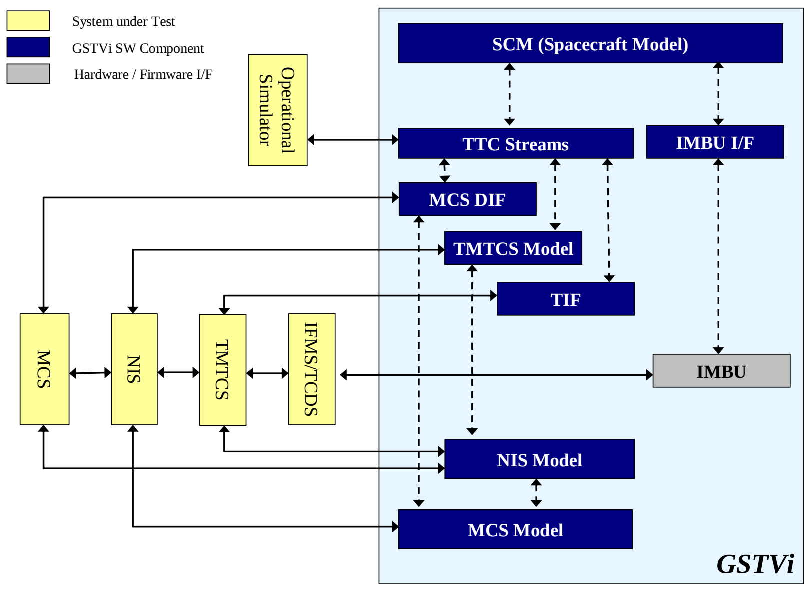

Fig. 1. Ground Systems Test and Validation Concept

The Ground Systems Test and Validation infrastructure (GSTVi) developed at the European Space Operations Centre (ESOC) provides a first implementation of the virtual space system concept. GSTVi consists of a set of simulation models that can be customized for specific missions and allow the simulation of the complete telecommand and telemetry chain from the Mission Control System (MCS) to the spacecraft and vice versa. The set of simulation models implementing this “virtual mission” includes an MCS model, a Network Interface System (NIS) model, models simulating ground station TM/TC processing equipment, and a spacecraft model. Depending on the test configuration to be supported, one or more real ground systems can be introduced into the chain while the others are replaced by their simulation model as illustrated in Figure 1.

All GSTVi components are fully configurable, and the way the components are used and connected within the simulation is also configurable and defined in “**Assembly**”.

GSTVI DESIGN AND IMPLEMENTATION

GSTVi was developed according to the Version 2 of the Simulation Model Portability Standard (SMP2), version 1.2, which defines a component model for development of simulation models as well as interfaces between the models and the runtime environment that allows the runtime to identify, load, configure and connect the models.
One particularity of the GSTVi software is that a large part of the software is based on existing software, which has been migrated in order to be SMP2 compliant.
Before looking into the software reuse, the development process chosen for GSTVi need to be introduced.

Development Process

For GSTVi, a “single source” approach has been used, where the design was performed using a UML tool and stored in a design database. The GSTVi documentation and source code has then been generated from this design database, which represented the “single source” for all phases of the project. This process ensured that design, implementation, and documentation have always been synchronous.

For GSTVi, the UML tool Magic Draw has been used. The reason why is that this tool has been used for the definition of both the SMP2 Standard and the EGOS Modelling Framework, and Magic Draw was the only tool where a profile for the EGOS Modelling Framework was available when the GSTVi project started.

The EGOS Modelling Framework (EGOSMF) has been a key tool used for the GSTVi development. The EGOSMF document generator (ADD, ICDs, DDD), catalogue generator (generation of SMP2 catalogue from the UML model), and C++ code generator (generation of C++ code from the SMP2 catalogue) have been used extensively during the project.

The development process has been performed according to the following phases:

- Initialisation: the GSTVi UML model has been initialised by importing existing UML design, and by performing code reverse engineering in order to import the design of existing code.
- Architectural design: the architectural design has been performed using the UML tool, with definition of all GSTVi components and external interfaces. The high level Software Design Document (Architectural) has been manually written using UML diagrams exported from the UML tool pasted into the document where appropriate. The component design has been fully generated from the design database. Dedicated Interface Control Documents have been provided for all external interfaces, again generated from the design database. This includes not only CORBA interfaces written in the Interface Definition Language (IDL), but as well SMP2 interfaces, an XML files. A document generator developed specifically by VEGA for the project has been used. As for the auto-documented design information, this ensures that the ICDs are always up-to-date, and that they are complete.
- Detail design and Implementation: the detail design has also been performed using the UML tool, and the C++ code generated from the model, with the exception of the GSTVi MMI components. Code has been generated using both the UML tool native code generator, and the EGOSMF catalogue to C++ code generator.

The SMP2 assemblies have been generated using the Framework Modelling Designer.

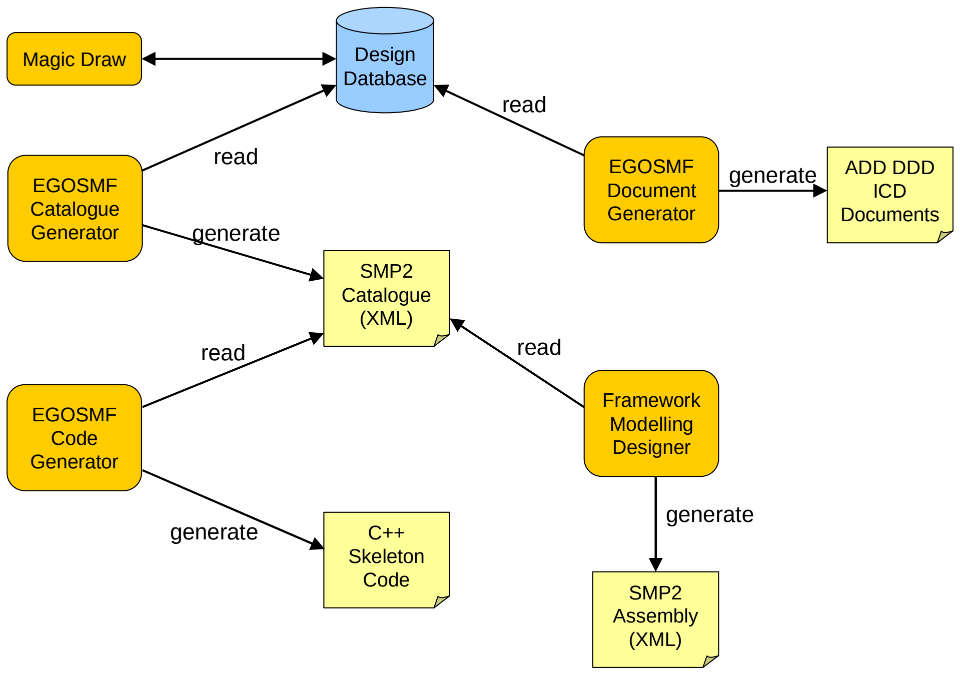

Fig. 2. Development Process

Runtime Environment

The GSTVi target run-time environment for GSTVi was SIMSAT 4.0. However, since SIMSAT4 was not yet available at the start of the project, it was decided to use a specific version of SIMSAT3 developed by VEGA at the start of the project. This version of SIMSAT3 included a SMP2 adapter compliant to the SMP2 standard version 1.2.

Integration with SIMSAT4 occurred only at the end of the GSTVi design and implementation phase. The full design and implementation phase, including the integration tests, has been performed on the enhanced version of SIMSAT3. During the integration with SIMSAT4, it was proven one of the main benefits of the SMP2 standard, namely the "portability". Integration of the GSTVi SMP2 models with SIMSAT4 was performed very efficiently and needed very little time. No modification of the GSTVi SMP2 models has been required there.
Moreover, it was proven that the SMP2 assembly and SMP2 catalogue files generated from the EGOSMF and the Framework Modelling Designer tools could be used directly with SIMSAT4 without any modification.

Software Reuse

In order to reduce the development time and cost, software has been reused for GSTVi as far as possible. The following software, or part of them, have been used for the GSTVi development:
- The Portable Spacecraft Simulator (PSS) for the implementation of the GSTVi spacecraft model, and the implementation of the IMBU interface;
- The NDIU Lite software for the implementation of the EGSE FEE I/F GSTVi model;
- The Hershel-Planck TM/TC Encoder and Decoder for the encoding and decoding of telemetry and telecommand packets in the GSTVi spacecraft and MCS models;
- The NCTRS for the implementation of the Network Interface Model (NIS).
- Some component from SIMPACK for the telemetry frame processor in the MCS model.
- Some component from the ROS/MEX simulator and Cryosat simulator for the implementation of PUS services on the GSTVI spacecraft model:
    - Memory management (PUS service 6)
    - On-board storage and retrieval (PUS service 15)

The work performed during the design and development phase on the reused software was to:
- Re-design the software architecture to "match" the SMP2 component architecture;
- Split some components into several SMP2 compliant models;
- Implements all interfaces to the simulation environment (SIMSAT4);
- Port the software to Linux platform. Some reused software component had been developed initially for the Windows or Solaris platform, and needed porting to Linux.

The most challenging part has been to re-design the software architecture of existing software components in order to be SMP2 compliant. Two possible approaches have been identified there:

<u>1st Approach:</u> the existing component is fully ported to SMP2. All the C++ classes of the component are converted to SMP2 models, and they use exclusively SMP2 native types and SMP2 interfaces. All the skeleton of the C++ code is generated from the SMP2 catalogue.

<u>2nd Approach:</u> the existing component is kept as is, and a SMP2 layer is added to the component to implement the required SMP2 interfaces, and to allow publication of the component fields and operations.

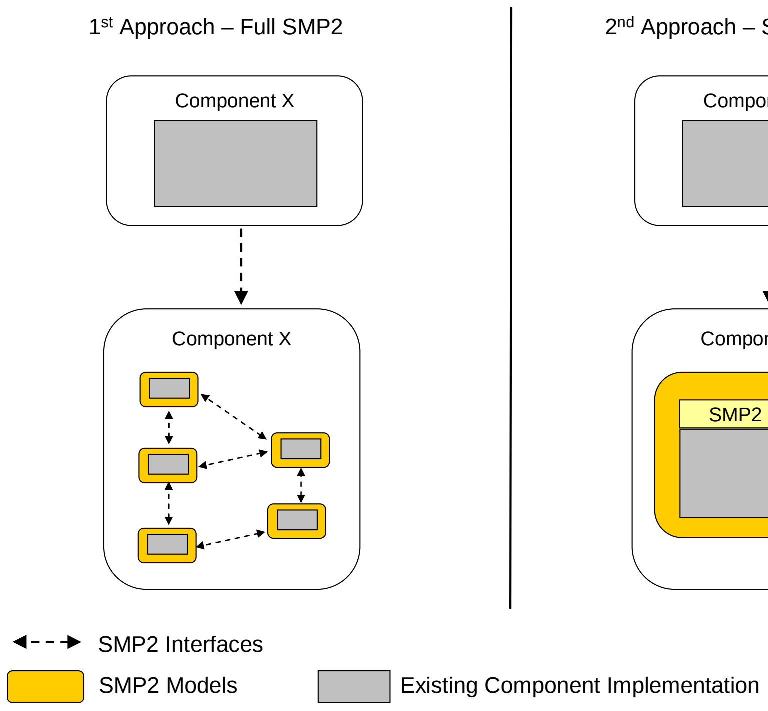

Fig. 3. SMP2 Migration Approaches

The two approaches have been used for GSTVi, depending on the amount of code to reuse, the type of component, and the feasibility of the approach. From this experience, the following advantages and drawbacks have been identified:

	1st Approach: Full SMP2	**2nd Approach: SMP2 layer**
Advantage	The component is fully SMP2, and can take full advantage of all SMP2 functions. Only one environment development needs to be available, i.e. the SMP2 compliant one with associated tools. The code is uniform and easier to maintain and understand.	The existing code can be re-used with a reduced effort and risk. There is no need to re-test all component functions, since most of them remain un-changed.
Drawback	The modification of the existing code to be fully SMP2 compliant requires a big effort, and requires full re-testing.	To interface the SMP2 code to the existing non-SMP2 one, additional code needs to be written which only purpose is to convert types/interfaces and to publish data. The code is not uniform. For instance, SMP2 code uses SMP2 types, the non-SMP2 code not. The development environment is not uniform across one component.

From a pure software design and implementation point of view, the 1st approach is the best since it eases the maintenance where only SMP2 code and the SMP2 development environment needs to be maintained, and offers better possibility for possible evolution of the software. However this approach requires big effort, which translates to accrued cost and development time, and is therefore not possible for reuse of large software components.

GSTVi Assembly

SMP2 assemblies define how a collection of SMP2 model instances is assembled for the runtime simulation, and offers the possibility to initialise SMP2 model fields. For GSTVi, several assemblies have been defined, where each assembly specifies typical combinations of GSTVi components in order to support specific test cases.

The concept of SMP2 assembly was not providing all needed functions for GSTVi. For instance, there was the need for GSTVi to allow definition of a script to execute when an assembly is loaded, and to allow linking an SMP2 assembly with a SIMSAT architecture file defining the components loaded in the simulation at runtime.

Therefore, we defined for GSTVi a GSTVi assembly which consists of:

- A SMP2 assembly file;
- A SIMSAT architecture file;
- A script file used to load the GSTVi model libraries or run a specific setup test.

The advantage of using the GSTVi assembly is that users only need to select this one from the MMI. Once the assembly is selected, the GSTVi Runtime Configuration Manager which control all GSTVi components then automatically:

- Selects the SIMSAT architecture file;
- Start the SIMSAT simulation;
- Execute the configured start-up script to load the GSTVi model libraries, load the catalogues, and perform other needed initialisation steps;
- Load the SMP2 assembly;
- Execute the configured setup script to perform any final initialisation steps.

GSTVI AND SMP2

Support for Loading and Publication of Sub-Models

For GSTVi, there is a need for a SMP2 model to dynamically create and publish a sub-hierarchy of models and associated data fields at runtine. This is needed to support dynamic loading of GSTVi mission tailoring databases at run-time, where the sub-hierarchy of a model (e.g. of the spacecraft model) is determined by the model specific configuration data (e.g. TM/TC parameters).

In the SMP2 version 1.1 specification, dynamic publication of sub-models was not possible as the `IPublication` interface did not support the self-publication of sub-models. This was due to the fact that normally the hierarchy is determined by the SMP2 Assembly and the model only publishes its data fields, and not contained models.

Therefore, for GSTVi, the SMP2 standard in its version 1.2 has been modified to allow this publication of sub-models.

Support for Dynamic Publication

The dynamic loading of GSTVi mission tailoring databases at run-time requires that additional data (and models) may be published during run-time and not only at initialisation time. However, the semantics of the SMP2 1.1 simulation states only ensured that all field values from the SMP2 Assembly have been set in *Configuring* state, which is after the *Publishing* state. This would therefore not allow GSTVi to publish the dynamic data read from the mission tailoring databases since GSTVi could only read the data *after* the field values from the Assembly have been set.

Therefore, for GSTVi, the semantics in SMP2 version 1.2 was changed to allow publishing of new data at any time during initialisation in *Building* state. This allowed to dynamically load new or updated configuration data during the initialisation phase.

However, this is still a limitation of the SMP2 standard, which imposed important constraints for the design and implementation of the GSTVi models.

The SMP2 standard allows for creation of models and publication of data fields only in *Building* state. This is fine for modelling of static models, which do not create any model or data during the simulation, but is an important limitation for modelling of models comporting dynamically created models/data.

For instance, GSTVi implements a Mission Control System which manages a command stacks and telecommands. The size of the command stack is not know at initialisation, nor it is known which telecommands will be created and dispatched. The only way to model this in SMP2 would be to create a pool of SMP2 telecommand models with predefined names, and at runtime to affect the dynamically created telecommands to the telecommand models from the pool. This mechanism complicates the source code, and has the drawback that at some point of time it may be that not enough models are created in the pool for the created telecommands. Moreover, in the simulation, it is nearly impossible to check the telecommands in the simulation tree or ANDs, since the SMP2 telecommand models have been created with predefined names and it is not known to which SMP2 model from the pool has been affected one telecommand.

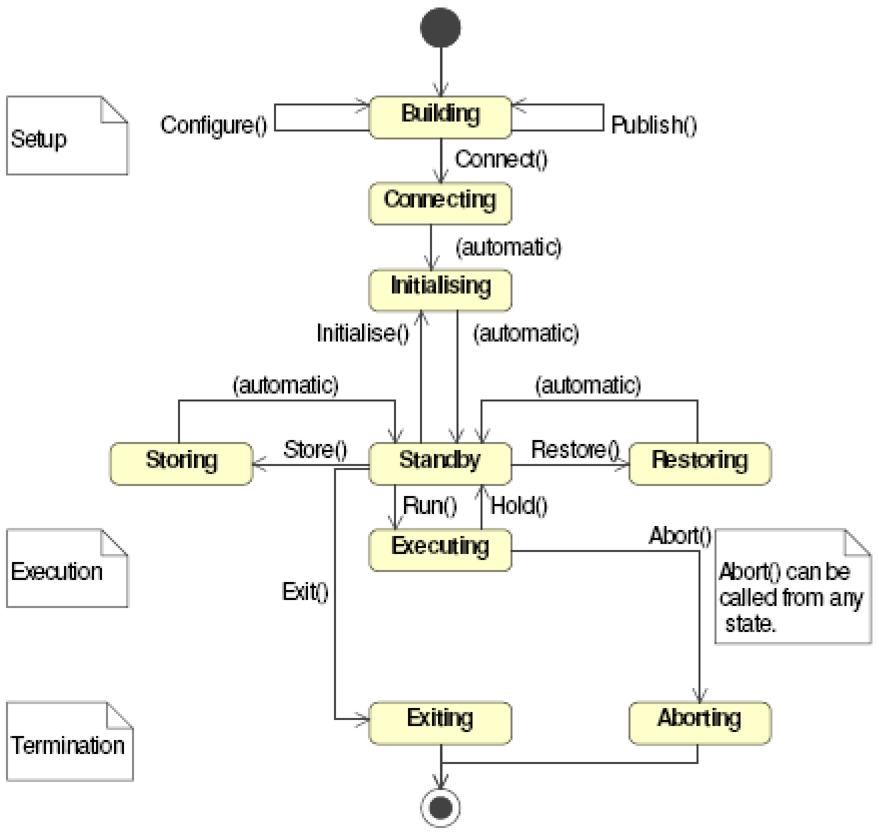

Fig. 4. Simulation Environment State Diagram

Moreover, for GSTVi, there was the requirement to allow support for simulation of several spacecraft in the same session. This feature was available in the Portable Spacecraft Simulator (PSS), the predecessor of the GSTVi spacecraft model. While the PSS was not required to run concurrent simulation of several spacecraft, it was required to allow switching between different spacecraft without the need to stop and restart the system. The spacecraft to load could be selected by the user at runtime.

Due to the constraints of SMP2 concerning the support for dynamic publication of models and data, it was not possible to directly achieve the same function implemented by the PSS. Once a spacecraft has been initialised and the simulation state changes to the *Standby* state, it is not possible with SMP2 to switch to another spacecraft since it is not possible to create and load new SMP2 models.

Therefore, for GSTVi, it was decided to re-start the entire simulation each time a new spacecraft is selected.

CONCLUSION

In this paper we have presented the Ground Systems Test and Validation Infrastructure concept and the practical experience on using the new SMP2 standard for the development of the GSTVi system. The paper describes how the SMP2 standard has been used, how the GSTVi system has been designed to be SMP2 compliant, the benefits of using SMP2 but also the constraint it imposes on the development process.

The main points can be summarized as follows:

GSTVi SMP2 Experience	
Support for Dynamic Simulation	SMP2 does not provide support for modelling of dynamic data. Once the simulation exits the *Building* state, it is not possible to create and load any new SMP2 model, and to publish additional data fields.
Support for unloading of models	SMP2 does not provide support unloading of SMP2 models. However, this is needed to fully support dynamic simulation, to limit the resources taken by the simulation.
Tools - reliability	The SMP2 development process relies on the development tools which take a very important place. Tools must be very very reliable, and easy to use and integrate in the development process.
Tools – concurrent development	Developing SMP2 models concurently is an issue, since not only the C++ code need to be shared, but also the UML model, the catalogues and assemblies. There are good tools to develop in C++ concurrently, but working in parallel on UML model or SMP2 catalogues is an issue.
Portability	It was proved during the GSTVi development that portability was not only one qualifier for SMP2, but was effective. The GSTVi models have been ported from SIMSAT3 to SIMSAT4 with very little effort.
Migration of code to SMP2	The migration of existing code to SMP2 is not an easy task, and can be very time consuming. Careful analysis on the existing code must be performed and clear decision taken to drive the design process.
SMP2 Events	The SMP2 standard allows to define events, to fire events, and to perform specific processing on event reception. However, one important missing function is that there is no way to associate data with the event. For GSTVi, we had the need to associate data together with an event, for instance, to provide the name of a telemetry parameter when the parameter value goes out-of-range. This was not possible directly using SMP2.

GSTVi SMP2 Experience	
SMP2 Interfaces	SMP2 allows definition of interfaces which are used by the models to interface to each-other, but does not provide any reference counting mechanism in order to control interface usage and allow proper object deletion. This implies that objects implementing interfaces defined in SMP2 are never deleted (i.e. are deleted only at the end of the simulation process), and that SMP2 models are never deleted.
Memory	SMP2 does not provide any mechanism for allocation and de-allocation of memory buffers exchanged between the SMP2 models. This means that problems may occur when porting the code to the Windows platform if there is a need for SMP2 models to exchange memory buffers
Loading shared library	SMP2 does not address how dynamic shared library implementing SMP2 models are loaded. No interface for loading shared library is provided. This is left to the simulation environment. This is however needed in case a SMP2 model needs to dynamically load and create a SMP2 model in the *Building* state.
Test automation	In SMP2, a model is represented by a single class. If a model implements several interfaces, all these interfaces must be defined for this class. The class may delegate the interface implementation to sub-classes defined in the SMP2 model. In case a SMP2 models supports may interfaces, lots of code must be written to delegate these interfaces which is time consuming and error prone.
GSTVi assembly	The GSTVi assembly proved to be a quite a good idea simplifying a lot the tasks of the end-users for starting the simulation, and providing the needed flexibility. Maybe some idea could be taken-up by the SMP2 standard.