

An experimental ECSS SMP development and runtime environment

Workshop on Simulation and EGSE for Space Programmes (SESP)
26 - 28 March 2019

ESA-ESTEC, Noordwijk, The Netherlands

Dr. Peter Fritzen⁽¹⁾, Dr. Stephan Kranz⁽¹⁾, Anthony Walsh⁽²⁾

⁽¹⁾Telespazio VEGA Deutschland GmbH

Europaplatz 5, 64293 Darmstadt, Germany

Email: peter.fritzen@telespazio-vega.de, stephan.kranz@telespazio-vega.de

⁽²⁾European Space Operations Centre

Robert-Bosch-Str. 5, 64293 Darmstadt, Germany

Email: anthony.walsh@esa.int

ABSTRACT

For many years, the SIMULUS Suite has provided support for SMP2 [1] Model Development (via EGOS-MF and UMF) and Execution (in SIMSAT). With the new ECSS SMP standard (E-ST-40-07) to be released soon [2], it is now time to move on and try out ECSS SMP in practice.

This paper summarizes the support that the latest SIMULUS 6.2 release provides for development and execution of models compliant with the public review version of ECSS SMP. While not the final version, this experimental implementation allows an assessment of the changes introduced by ECSS SMP, and a first familiarization with the new features provided by it. For this, UMF has been enhanced to support generation of ECSS SMP compliant source code, in addition to the continued support for SMP2. In the runtime, a new ECSS Adapter has been added to SIMSAT, capable of loading and executing models compliant with ECSS SMP.

While future models may be developed natively for ECSS SMP, it has to be considered that a large code base (e.g. at ESOC) exists based on the SMP2 standard, as released in 2005. Therefore, a second focus of the paper is on the migration of existing design and source code from SMP2 to ECSS SMP, and some first results and lessons learned for the Generic Models suite that is provided as part of SIMULUS. Finally, the paper explains how core parts of SIMULUS, namely the SIMSAT simulation services, can now be developed with its own development environment – a feature introduced by ECSS SMP by adding support for modelling of Services (in addition to Models).

INTRODUCTION

The SIMULUS Suite developed and maintained at ESOC provides support both for the development and for the execution of models compliant with the SMP2 standard. UMF supports a UML Model Driven Process:

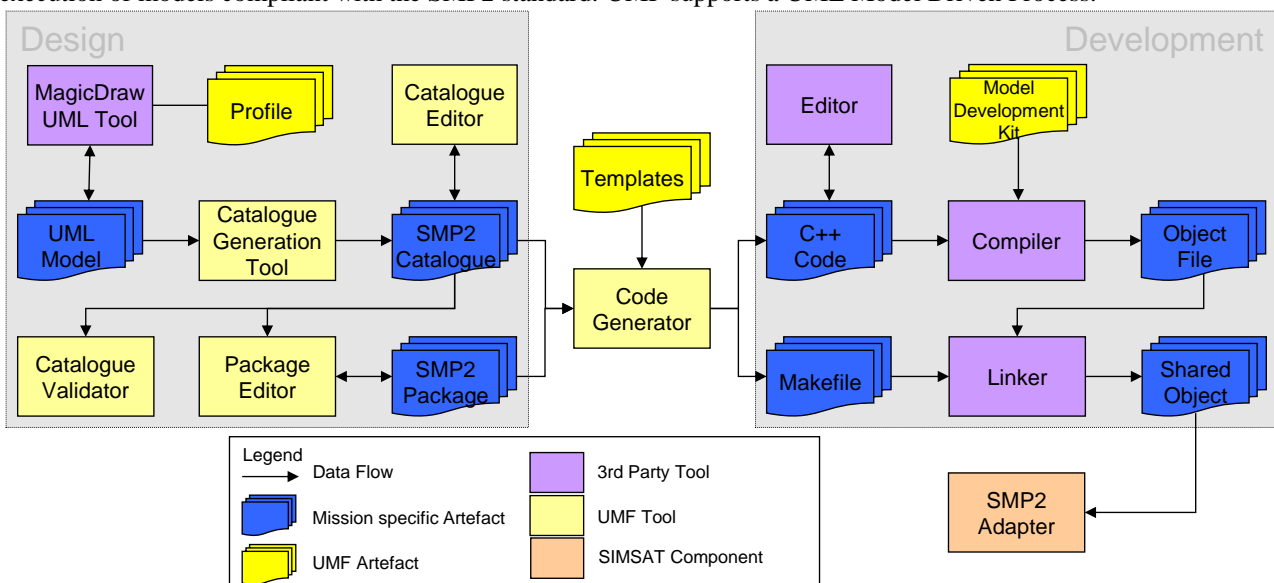


Figure 1 - Model Driven Software Design and Development Process of UMF

Within this process, the following UMF Artefacts and Tools have a dependency on the SMP2 Standard:

1. **Profile:** The UML Profile provides stereotypes that define a domain specific language for **SMP2**.
2. **Catalogue Generation Tool:** This generator converts the UML model into an **SMP2 Catalogue**.
3. **Catalogue Validator:** This validator is specific to the semantics of an **SMP2 Catalogue**.
4. **Catalogue Editor:** This editor is specific to the **SMP2 Metamodel**.
5. **Package Generation Tool:** This generator converts the UML model into an **SMP2 Package**.
6. **Package Editor:** This editor is specific to the **SMP2 Metamodel**.
7. **Code Generator:** This tool generates code for an **SMP2 Catalogue** or **SMP2 Package** using **Templates**.
8. **Templates:** The templates of the Code Generator have a dependency on the **Model Development Kit**.
9. **Model Development Kit:** This library of C++ classes implements the **SMP2 Component Model**.
10. **SMP2 Adapter:** This SIMSAT component can load and execute **SMP2 Models**.

APPROACH TO ADD ECSS SMP SUPPORT TO UMF

To add ECSS SMP support to UMF, without losing support for SMP2, all these artefacts and tools have to be provided in a second flavour suitable for the ECSS SMP standard. While the **tools** (generators, validators and editors) have been developed as clones of the SMP2 versions binding them to an Eclipse Modelling Framework (EMF) model of ECSS SMP rather than SMP2, a different approach has been used for the artefacts.

- The **Profile** has been extended by additional stereotypes so that it supports both SMP2 and ECSS SMP at the same time. Examples of such changes include a new stereotype for `<<SMP2service>>`, or an additional unit argument for the existing `<<SMP2integer>>` stereotype.¹
- The **Model Development Kit** has been replaced by a **Component Development Kit** which supports both Models and Services. This decision was driven both by the new support for Services in the Metamodel, and for the decision to use UMF and design and development environment for SIMSAT Next Generation [3].
- The Code Generator **Templates** for ECSS SMP are based on those for SMP2, but now generate code that makes use of the Component Development Kit. Some further clean-up has been done, partially driven by changes in the standard, partially taking the opportunity to update the templates to generate C++ 11 code.

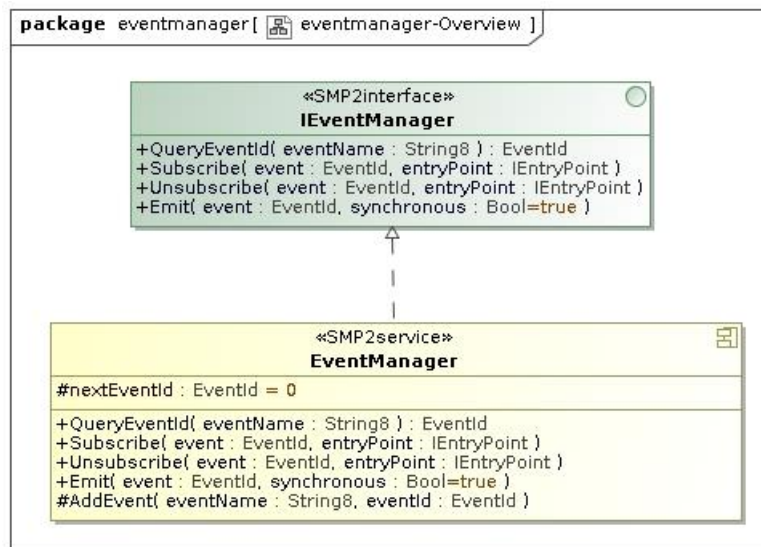


Figure 2 - Example of an SMP Service modelled using the new `<<SMP2service>>` stereotype

Command Line Support

When using UMF on the Command Line, typically from a Build System making use of maven or make, support for both SMP2 and ECSS SMP in a single installation of UMF is provided by an additional command line argument (`--standard`). During the migration of the Generic Models, it was hence possible to generate both SMP2 catalogue and code and ECSS SMP catalogue and code from the identical UML design, using the identical UMF version, but different command line parameters. All build commands are maintained unchanged. This significantly eases the migration process, as any change can immediately be tested on both standards. An example for code generation is provided below.

```

esa.umf.cli.UmfClient --server --standard ECSS-SMP --timeout 120 generate
--artefact code --output Src/esa.simsat.services.eventmanager/generated/code
Src/esa.simsat.services.eventmanager/umf/esa.simsat.services.eventmanager.smpcat
  
```

¹ For the sake of backwards compatibility, the use of “SMP2” in the stereotypes has been maintained, although “SMP” would be the correct acronym to use for ECSS SMP.

User Interface Support

From the Eclipse based User Interface, each UML solution has an additional parameter that can be selected during setup, and that defines whether the solution (and hence all projects in this solution) use SMP2 or ECSS SMP. Examples of a UML solution are the Generic Models, the Reference Architecture, or an Operational Mission Simulator (TOMS). It is assumed that all projects within one solution use the same standard, which is a sensible assumption (as SMP2 and ECSS SMP cannot be used in one process space, due to namespace conflicts). The individual tools look identical for both standards, and are started the same way, automatically taking into account the solution settings. Again, after switching from one standard to the other standard, no additional changes in the use of UMF are needed.

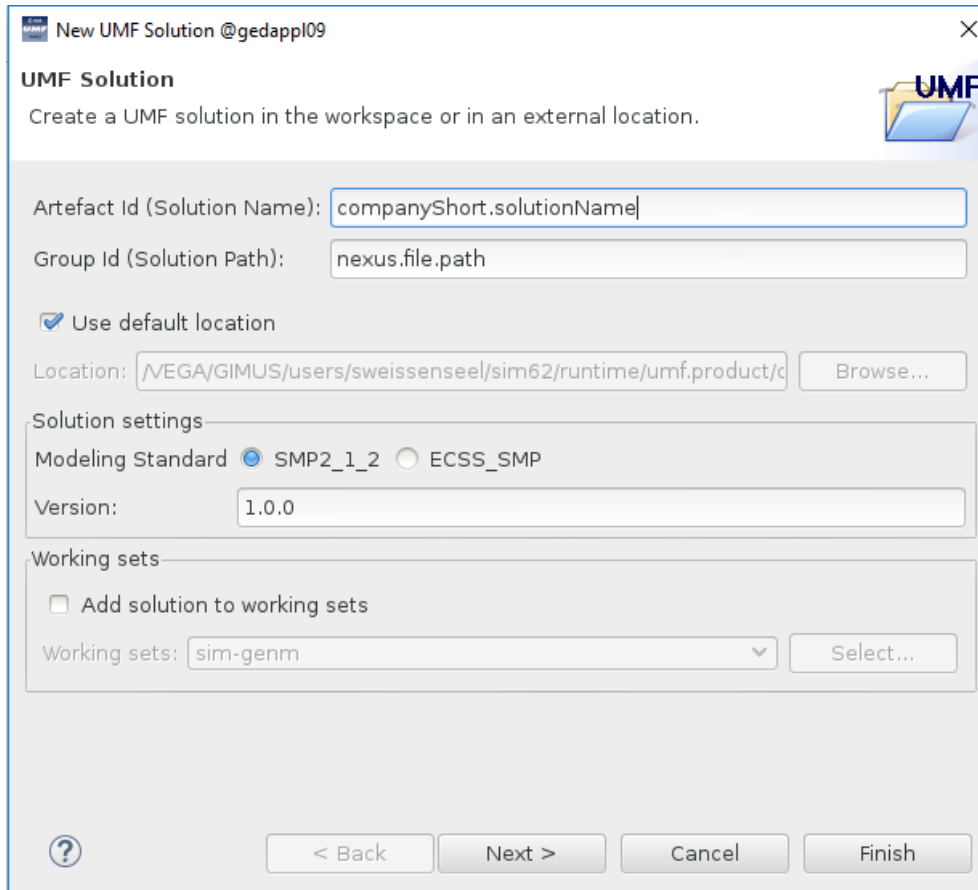


Figure 3 - A UMF Solution can select between SMP2_1_2 and ECSS_SMP

Development of new Models

With the latest version of UMF (UMF 3.2 from SIMULUS 6.2), the development of models for either of the two simulation standards is almost identical. Some care has to be taken when developing SMP2 models; the new features added to the Profile have no effect and are ignored by the Catalogue and/or Code Generator, as they are not supported in SMP2. When developing models for the new ECSS SMP standard, all features can be used, as the ECSS SMP Catalogue format is a superset of SMP2. No major issues are expected for new developments.

Migration of existing Models

ECSS SMP was integrated making an effort to minimise migration effort for existing models. However, difficulties are to be expected in the few areas where ECSS SMP and (the UMF/SIMSAT tailoring of) SMP2 have a different semantic. Two examples of such changes are forcing and failing, which were not covered by SMP2. As an extension, this was added to UMF and SIMSAT, but in a way not fully compatible with the interfaces now defined by ECSS SMP. Therefore, use of these features requires a different approach by the modeller (change of semantic), and hence design and code changes to existing models. Further changes are introduced by new features, such as support for modelling of Services and Exceptions, which now can (and should) be done according to the ECSS SMP standard.

Finally, the Component Model has been consolidated in ECSS SMP, leading to the removal of several interfaces (e.g. all interfaces in the namespace `Smp/Management`). In rare cases, where models explicitly use such interfaces (rather than the code generated by the Code Generator), changes to source code (or even UML design) are necessary.

Therefore, a migration of existing Models (UML design and source code) can either be done only to a level that the code compiles and runs (with catalogue and code generation done by UMF), or to a level where all new features added to ECSS SMP are actively used. Clearly, the latter approach is more effort, but ensures that features like forcing and failing are used in a generic (standardised) way, and hence will work outside of SIMSAT as well.

MIGRATION OF THE GENERIC MODELS

As an exercise to assess the impact of the ECSS SMP standard on existing models, and as well to validate the experimental implementation of ECSS SMP in SIMULUS 6.2, the Generic Models (GENM) of SIMULUS have been migrated. Due to time and budget constraints, only an initial migration was done, allowing to compile the code versus ECSS SMP, and to run all automated unit and integration tests. As much as possible, the migration has been automated using scripts, but manual changes were needed in several places. The steps and findings of the migration are summarized below.

Migration of UML Models

As a first step, the dependency of the UML model on SMP2 has to be replaced by a dependency on ECSS SMP. This has various implications:

- Each use of an SMP2 type (AnySimple, EventId, etc.) has to be replaced by the ECSS SMP counterpart
- Each use of an SMP2 interface (IObject, IModel, etc.) has to be replaced by the ECSS SMP counterpart
- Each use of an SMP2 attribute (Operator, Minimum, etc.) has to be replaced by the ECSS SMP counterpart

For GENM, it was possible to fully automate a). For b), manual changes were needed, but this was rarely used. Significant effort was the migration of SMP2 attributes, due to the decision taken in earlier versions of UMF to add UMF specific attributes under SMP2 (in namespace SMP/Attributes). Some of these attributes are now standardized as part of ECSS SMP, and hence can be re-linked to their counterpart. For those not standardized, a new extensions namespace was created under esa::ecss::smp, to clearly document that these attributes are part of an ESA extension of ECSS SMP, and are not part of the standard. The UML model providing these attributes has to be referenced.

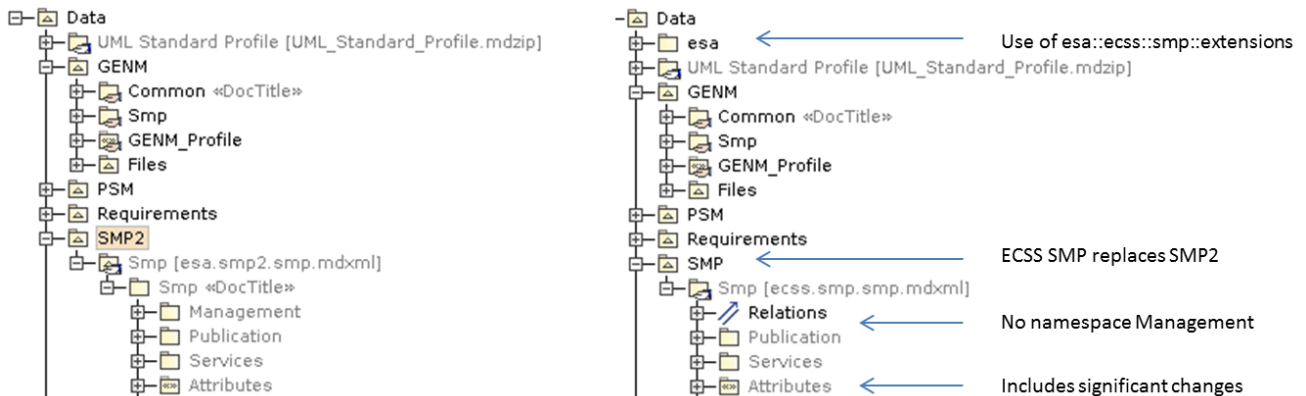


Figure 4 - UML Design of of esa.genm.common using SMP2 (left) and ECSS SMP (right)

Migration of Source Code

A significant part of the GENM source code was generated by the UMF SMP2 Code Generator. As the approach for code generation has not been changed, the first step of the migration is a re-generation using the new ECSS SMP Code Generator (and the integrated code merger). In the ideal case, this results in code that immediately compiles. In reality, this had been the case for only very few of the about 30 source code projects, while the majority of code needed further changes. As a second step, fully automated by sed scripts, changes to SMP namespaces were applied. The table below is not complete, but names a few typical replacements in code.

Table 1 - Examples of code changes required due to differences between SMP2 and ECSS SMP

Old SMP2 Code	New ECSS SMP Code
Smp::IDynamicSimulator	Smp::ISimulator
Smp::Management::IManagedObject	Smp::IObject
Smp::MKS_Created	Smp::ComponentStateKind::CSK_Created
Smp::Services::LMK_Information	Smp::Services::ILogger::LMK_Information
SimpleTypes	PrimitiveTypes
Smp::ST_Int32	Smp::PrimitiveTypeKind::PTK_Int32
Smp::Publication::Uuid_Int32	Smp::Uuids::Uuid_Int32

These changes mostly result from three different sources:

- In ECSS SMP, the interfaces have been rationalized, and many interfaces have been merged.
- In C++ 11, enumerations can now be represented as strong types, allowing fully qualified literals.
- In C++ 11, constants shall be defined within classes, not within namespaces.

While the number of source code lines affected by such changes was not marginal, the effort for the migration was minor, due to the automation via scripts. The same scripts had been applied on test code later.

Manual changes had to be made in several places, including the following:

- Manual publication: Where fields or operations are published manually (i.e. not in code that was auto-generated by the UMF Code Generator from a Catalogue), the `view` flag (in SMP2) had to be replaced by a `view` state (from an enumeration, e.g. `Smp::ViewKind::VK_All`)
- Use of `GetParent()`: This operation returns `IComponent` in SMP2, but `IObject` in ECSS SMP.

The above listed changes were necessary to compile existing source code versus the ECSS SMP standard (and the new Component Development Kit, CDK, developed as part of SIMULUS Next Generation). Changes required due to the change from `Smp::Mdk` to `esa::ecss::smp::cdk` are not reported here, as it was an explicit design decision to change the namespace; an ECSS SMP MDK can well be kept in the namespace `Smp::Mdk`. The majority of such changes are automated as well, and only few manual changes were needed.

While most of the models did immediately pass their unit tests, some failed because of problems in the source code not noticed in SMP2. A few typical reasons for such failures are reported below.

Storage of Fields in a Collection

In ECSS SMP, a new interface `IField` and a corresponding operation `GetFields()` of the `IComponent` interface have been added. Correspondingly, the fields of a model are now stored in a collection, which mandates unique names. While unique names for the fields of a model are a sensible idea in SMP2 already, the SMP2 Model Development Kit does neither check nor enforce this. Therefore, several models do exist in GENM which publish the same field twice (not a real problem, but a memory overhead), or even two different fields with the same name (in which case only one of them is accessible). In all these cases, ECSS SMP now terminates during publication with a `DuplicateName` exception.

MIGRATION OF GENERIC MODELS TESTS

Specifically for the SIMULUS Generic Models, it can be reported that the migration of test code took more effort than the migrated of UML design and model source code. This is partially caused by the fact that test code is hand-written and does not benefit from automated generation by a code generator. In addition, a few changes in the standard caused more severe changes to the tests. While this may be specific to GENM and the design of unit and integration test code, problems encountered several times are reported here, as they may be found in other model libraries or simulators as well.

Use of Stub Models for Testing

Many of the Generic Models unit or integration tests use stub models. While these are SMP2 models that could (and maybe should) have been generated from a UML design and SMP2 Catalogue, they had typically been hand-written. This has significantly increased the effort of their migration from SMP2 to ECSS SMP, especially for publication (all methods of the `IPublication` interface have changed) and state transitions (the `Configure()` operation has a new parameter in ECSS SMP). For future maintenance of the stubs, it is therefore recommended to generate them from UML design using UMF.

Assembly from Source Code

GENM Unit Tests typically test a single class, model or service, while integration tests typically test a small assembly. This assembly is generated with C++ code. The difficulty is now caused by the fact that ECSS SMP models need to get their parent provided at construction time (in the constructor), no longer when they are added to a container of their parent; in SMP2, the parent was updated using `SetParent()` when adding a model under its parent model, but this setter has been removed in ECSS SMP. In the existing test code, it was found that constructors were often called without a parent (which was easy to spot and fix), but in some cases with a wrong parent (which was later fixed using the `SetParent()` call). These cases were harder to spot, so a cross-checking in the `AddComponent()` operation of the `Container` template has been added.

REFERENCES

- [1] SMP2 Standard: SMP2 Handbook (EGOS-SIM-GEN-TN-0099), SMP2 Metamodel (EGOS-SIM-GEN-TN-0100), SMP2 Component Model (EGOS-SIM-GEN-TN-0101) and SMP 2.0 C++ Mapping (EGOS-SIM-GEN-TN-0102), all Issue 1 Revision 2 from 28 October 2005
- [2] ECSS-E-ST-40-07C DIR1: draft distributed to the ECSS community for Public Review, 23 October 2018
- [3] Designing a Simulation Environment fit for the next ten years, Alberto Ingenito, Peter Fritzen, Paul Steele, James Eggleston, SESP 2019