An Industrial Validation of SMP

Hien Thong Pham[1], Rachid Atori[2], Bernard Delatte[3], Claude Cazenave[4], Gilles Mesiano[5], Pierre Dissaux[6]

[1] Spacebel
I.Vandammestraat 7, 1560 Hoeilaart, BELGIUM
Email: hienthong.pham@spacebel.be

[2] Spacebel
I.Vandammestraat 7, 1560 Hoeilaart, BELGIUM
Email: rachid.atori@spacebel.be

[3] Centre National d'Etudes Spatiales (CNES)
18, Avenue Edouard Belin, 31401 Toulouse, Cedex 9, FRANCE
Email: bernard.delatte@cnes.fr

[4] Astrium SAS
31 Rue des Cosmonautes, Z.I. du Palays, 31402 Toulouse Cedex, FRANCE
Email: claude.cazenave@astrium.eads.net

[5] Thales Alenia Space
100 Boulevard du Midi, 06156 Cannes La Bocca, FRANCE
Email: gilles.mesiano@thalesaleniaspace.com

[6] Ellidiss Technologies
24, Quai de la Douane, 29200 Brest, FRANCE
Email: pierre.dissaux@ellidiss.com

INTRODUCTION

The E-40-07 SMP2 standard aims at providing a framework that allows simulation model portability across the simulation infrastructures that exist in the different simulation companies. Prior to a wide adoption by the industry, the concepts promoted by the SMP2 standard must be proven, or in other words validated with an industrial strength process. The SMP2 support tools for models design and code generation need to be tested with real simulation cases.

"Validation Industrielle de SMP2 (VISMP2)" is an initiative of the CNES (Prime) with the cooperation of Spacebel, Thales Alenia Space, EADS Astrium and Ellidiss Technologies (Participants). The study aims at validating the SMP2 standard through the design of a simulator following an industrial process, which involves several major actors of the space simulation industry. Lessons learned and recommendations will be provided as input to the ECSS. The objectives of the study were settled as follows:

- Design of a realistic SMP2 simulator.
- Distributed development (or multi-sites).
- Validation of the user needs from the use of SMP2 tools.
- Validation on multiple simulator infrastructures.

The SMP2 simulator will be a FES (Functional Equipment Simulator) type simulator of a representative spacecraft model, including the Space Segment, the Ground Segment and the Environment.
The SMP2 tools to exercise will be SIMSAT4 MIE (ESOC) and STOOD (Ellidiss). SIMSAT4 MIE will be the main supporting tool. STOOD, which provides a design graphical interface while SIMSAT4 does not, will be used to generate catalogues and code for comparison with SIMSAT4.
The distribution of the development will allow each participant to take different roles in the process and to share information (SMP2 catalogues, assemblies, models source and binary code).
The final simulator will be validated on the SIMSAT4 (ESOC), BASILES Kernel (CNES/Spacebel), and SIMTG (Astrium) simulation infrastructures. It is not explicitly stated further in the article but the three platforms are obviously based on similar hardware and OS to make the exchange of binary code possible.

The SMP2 version to be considered in this study is the v1.2 of the standard.

PROCESS DESCRIPTION

Like in a classical industrial process, there are 3 main phases:

- System Architecture Design: in this phase, the overall system architecture is specified. The models, the interfaces between the models and the models hierarchy are the output of this phase.
- Models Development & Test: in this phase, models are implemented and tested.
- System Integration & Validation: in this phase, all models are integrated to make the final simulator assembly. The simulator is validated on the various simulation platforms. Problems are corrected if necessary.

Each of the participants in the study has a specific role during each phase of the project. Table Tab. 1 shows the respective roles of Spacebel, Thales, Astrium and Ellidiss.

Tab. 1. Participant roles in the process

Phase	Role	Responsible Participant
System Architecture Design	Main	Spacebel
	Contributors	Thales Astrium CNES
Models Development & Test	Main	Spacebel Thales Astrium
System Integration & Validation (SIMSAT4, BASILES Kernel)	Main	Spacebel
	Contributors	Thales Astrium
System Integration & Validation (SIMTG)	Main	Astrium
STOOD support	Main	Ellidiss

Simulator

Fig. 1 shows the high-level simulator architecture targeted by the VISMP2 activity. This is a representative satellite simulator including the Space Segment, the Ground Segment and the Environment.

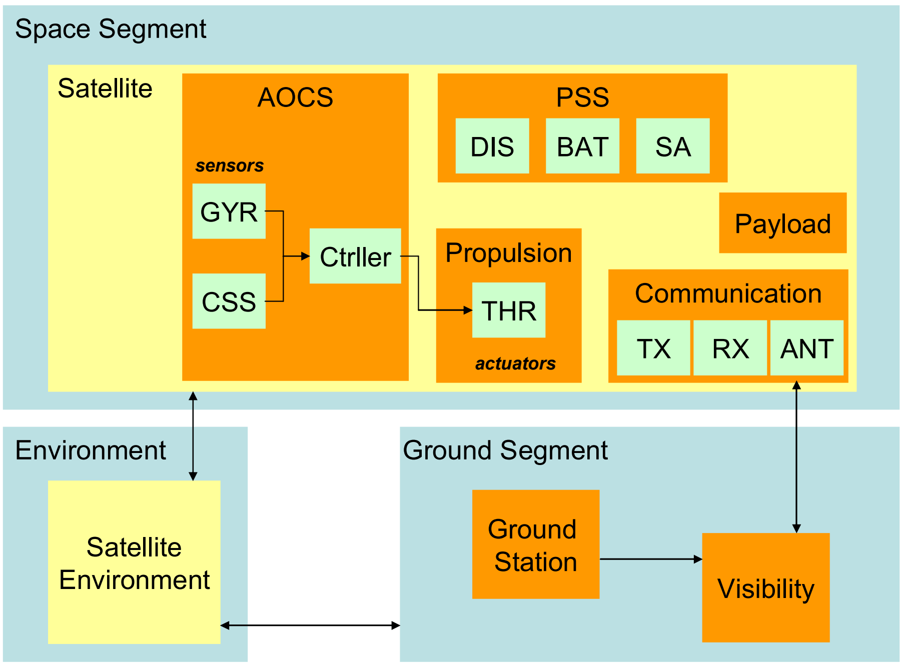


Fig. 1. VISMP2 simulator architecture

System Architecture Design

The System Architecture Design phase involves all participants.

The main responsible of the system architecture design phase proposes a first sketch of the simulator architecture. The architecture is expressed in a UML (Unified Modeling Language) model. This was created with the StarUML tool, which is freely available on the Internet. The starting point was the SIMVIS models library, which is simplified and extended wherever required. The AOCS and Environment models are provided by Thales and Astrium reusing existing models.

The UML model is discussed and refined by all contributors to come to a final version. As said previously, the UML model contains: 1) the common Interfaces, 2) the simulation models including their public Fields and Implemented Interfaces, 3) their SMP2 interfaces for inter-model communication (mostly Interface Links and Field Links, no Event Links) and 4) their hierarchy (inheritance, composition and aggregation). From the UML model, work can be distributed among all the participants (Tab. 2).

Tab. 2. Work distribution

Sub-system or Model	**Type**	**Responsible Participant**
AOCS	Sub-system	Thales
AOCS Controller	Model	Astrium
PSS	Sub-system	Astrium
Solar Array	Model	Spacebel
Communication	Sub-system	Spacebel
Transmitter	Model	Thales
Propulsion	Sub-system	Thales
Payload (simplified)	Sub-system	Spacebel
Ground Segment (including Visibility)	Sub-system	Spacebel
Common	Common Interfaces and top-level Models	Spacebel
Assembly File (Simulation Architecture)	SMP2 file	Spacebel
Schedule File	SMP2 file	Spacebel

Fig. 2 shows the involvement of each participant in the simulator architecture.

As shown, the work distribution aims at promoting as much as possible the exchange of information between the participants. Each will have the opportunity to make at least a sub-system (development & integration role) and one model from another sub-system under another participant responsibility (development only role).

The UML model contains as well a collaboration diagram that shows interaction between model instances in the simulator and scheduling sequence. This is the basis for building the SMP2 Assembly and Schedule.

In the next step, SMP2 catalogues are created from the UML model. The tool used in this step is the SIMSAT4 MIE (Model Integration Environment). The UML to SMP2 catalogues mapping is a one-to-one mapping, i.e. no additional information is added.

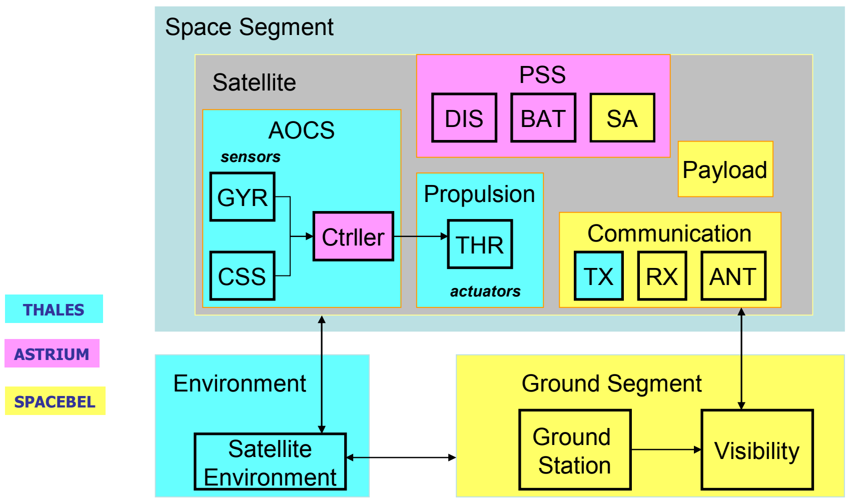


Fig. 2. VISMP2 simulator architecture with work distribution

In order to take into consideration the multi-sites and work distribution aspects, the catalogues are organized as follows:

- A common catalogue *(common.cat)* is created to contain the common models and interfaces between the models.
- Specific catalogues are created. There is one catalogue per participant and per sub-system of the satellite under the participant responsibility *(aocs_Astrium.cat, aocs_Thales.cat, com_Thales.cat, com_Spb.cat, ground_Spb.cat, pss_Astrium.cat, pss_Spb.cat).*

The common catalogue contains the interfaces that the models must respect so that the connection with other models is possible. This catalogue is not meant to be modified by a participant. A specific catalogue contains the first version of the models under responsibility of a participant. Each specific catalogue is further completed or refined at the participant site in the next phase of the process.

Models Development & Test

The Models Development and Test phase is realized at each participant site, using its own development environment.

During this phase, models specifications are further refined by completing the specific catalogue files. During this phase, changes to the common catalogue should be avoided as much as possible. If required, it must be discussed and agreed on with all the participants. Once the models design is stable, they can be implemented, i.e. coded in C++ in this case.

When the models implementation is terminated, these are tested in the same development environment. For this task, stub models must be created to verify the external interfaces of the models. The instances of the stub models and of the models under tests define together a SMP2 Assembly, called the *sub-assembly* as this corresponds to a subdivision of the SMP2 Assembly that covers the final simulator. In general, a participant will create a sub-assembly per sub-system and per isolated model under its responsibility. The sub-assembly has also a documenting role as it shows to the integrator how models of a sub-system are connected to other sub-systems. Models are packaged for execution in .so files, which correspond to the C++ mapping of SMP2 Packages.

The output of this phase, as delivered by each participant for integration, will comprise the following items:

- Completed specific catalogues.
- Models source code and binary code (.so file).
- Sub-assemblies served for testing the models.

System Integration & Validation

In this phase, the work from all participants is integrated together to build the final simulator, which can then be validated on various simulation platforms. The responsible of a sub-system (AOCS, PSS, Propulsion…) performs first an integration of the "missing" models of its sub-system, which have been developed by another participant. This is as agreed at the work distribution time.

Sub-System Integration & Validation

The sub-system integrator receives the external models delivered by another participant. In the corresponding sub-assembly, these will replace the related stub models. Then, validation is performed with the resulting sub-assembly. This step allows resolving possible problems or issues at sub-system level before the start of the system integration phase.

System Integration & Validation

The system integrator receives the deliverables from all participants (see the list above). A global SMP2 Assembly needs to be built by manually combining all sub-assemblies. Stub model instances are removed and replaced by real model instances. This step is done manually because no tool support is available. This can be partly explained because the sub-assembly is a notion that does not exist at that time in the SMP2 standard. Fig. 3 shows a high-level view of the simulator Assembly. The integrator makes a Schedule file according to the scheduling requirements of every model.

Two test configurations will be exercised: one using the delivered model source code and one using the delivered binary code. In the first case, a build environment needs to be setup to create the necessary .so files.

When the Assembly, the Schedule and Packages are ready, the simulator can be created and executed in the target infrastructures.

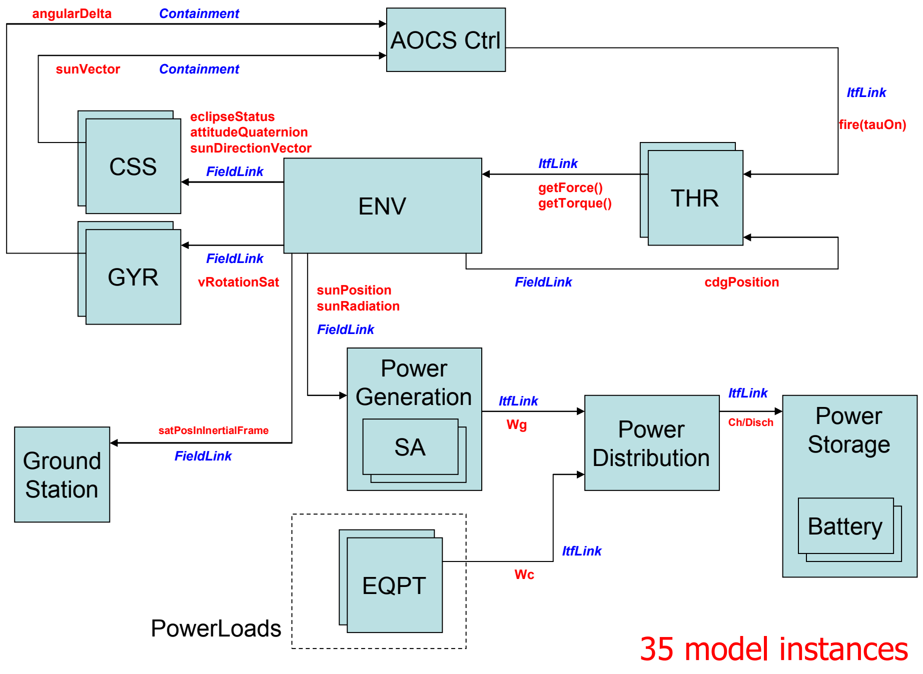


Fig. 3. VISMP2 Assembly Overview

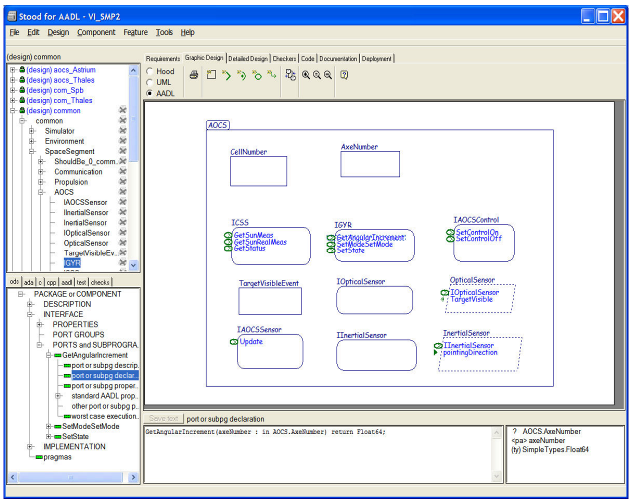
Process Recursion

It is to be noted that the process described above can be recursively followed for the design, implementation and test of a sub-system.

Design with STOOD

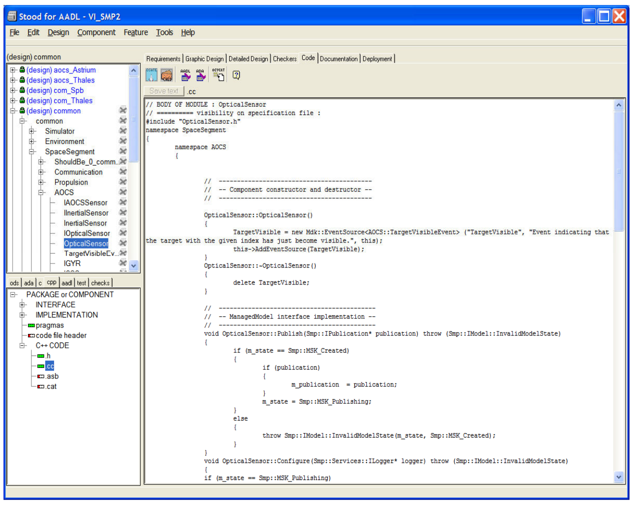
STOOD is a graphical design tool that is based on the AADL framework. A SMP2 adapter has been provided to support generation of SMP2 catalogues, assemblies and code. The tool provides a mapping of SMP2 notions over AADL. STOOD has the advantage over the SIMSAT4 MIE because it provides a graphical interface that allows the designer to have a better visualization of the system. SIMSAT4 MIE only provides a tree view.

In the VISMP2 study, the modeling is also done in STOOD in order to generate the SMP2 artifacts for the purpose of comparison with the ones obtained via SIMSAT4 MIE. This activity was done in parallel with the development process described above.



```
GetAngularIncrement(axeNumber : in AOCS.AxeNumber) return Float64;
```

```
? AOCS.AxeNumber
<pa> axeNumber
(ty) SimpleTypes.Float64
```



```
// BODY OF MODULE : OpticalSensor
// ========== visibility on specification file :
#include "OpticalSensor.h"
namespace SpaceSegment
{
        namespace AOCS
        {


                //  ------------------------------------------
                //  -- Component constructor and destructor --
                //  ------------------------------------------

                OpticalSensor::OpticalSensor()
                {
                        TargetVisible = new Mdk::EventSource<AOCS::TargetVisibleEvent> ("TargetVisible", "Event indicating that
the target with the given index has just become visible.", this);
                        this->AddEventSource(TargetVisible);
                }
                OpticalSensor::~OpticalSensor()
                {
                        delete TargetVisible;
                }

                //  -------------------------------------------
                //  -- ManagedModel interface implementation --
                //  -------------------------------------------
                void OpticalSensor::Publish(Smp::IPublication* publication) throw (Smp::IModel::InvalidModelState)
                {
                        if (m_state == Smp::MSK_Created)
                        {
                                if (publication)
                                {
                                        m_publication  = publication;
                                }
                                m_state = Smp::MSK_Publishing;
                        }
                        else
                        {
                                throw Smp::IModel::InvalidModelState(m_state, Smp::MSK_Created);
                        }
                }
                void OpticalSensor::Configure(Smp::Services::ILogger* logger) throw (Smp::IModel::InvalidModelState)
                {
                if (m_state == Smp::MSK_Publishing)
```

Fig. 4. STOOD Graphical Design & Code Generation

RESULTS

The simulation built with the models source code could run successfully on the three target infrastructures (SIMSAT4, BASILES Kernel and SIMTG).

Unfortunately, the simulation built with the models binary code could not run successfully on the three target infrastructures (SIMSAT, BASILES Kernel and SIMTG). The reason lies in the platform dependent way that each participant links a .so file (or SMP2 Package) with the Model Development Kit (MDK) library. This aspect shall be further investigated in future studies to address the binary compatibility.

The biggest achievement was to obtain AOCS convergence considering the differences in the nature of the AOCS Controller model (provided by Astrium) and the Environment model (provided by Thales). This is shown in the following figures (satellite rotation rate vector tend to 0, sun vector and solar array current tend to constant values).

The parallel design activities with STOOD provide comparable SMP2 artifacts with the reference ones obtained with the SIMSAT MIE.

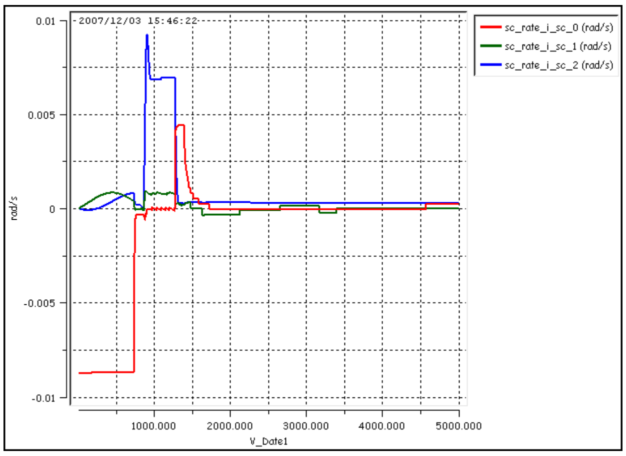


Fig. 5. Satellite rotation velocity

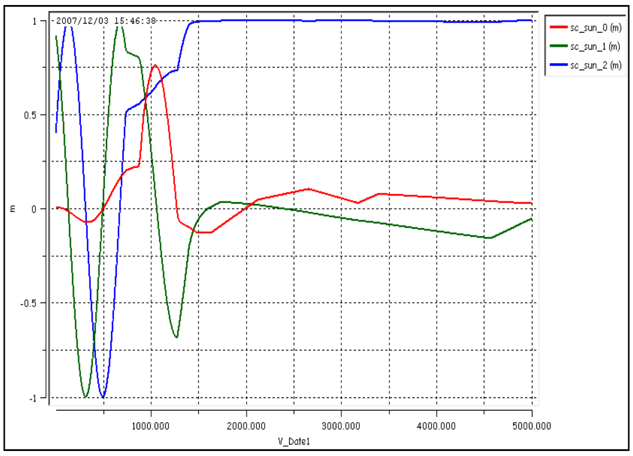


Fig. 6. Sun vector in the satellite frame

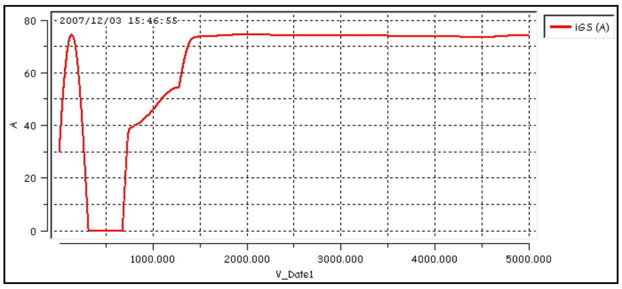


Fig. 7. Solar array generated current

CONCLUSIONS & LESSONS LEARNED

VISMP2 allows demonstrating a cooperative and efficient simulator development thanks to the SMP2 standard. Conclusions and lessons learned from this activity are detailed here after.

SMP2 vs. Process

From the process point of view, the VISMP2 activity shows where the SMP2 standard is the most profitable to designers of space simulation systems.

- System Architecture Design: in this phase, system designers will obviously need the support of a non SMP2 tool (UML design tool, STOOD…).
- Models Development & Test: in this phase, throughout the VISMP2 experience with multi-sites development, it can be observed that the SMP2 standard brings the real benefits. SMP2 serves as a common language between all development parties.
- System Integration & Validation: there is little support from SMP2 in this area. Sub-assemblies appear to be very useful for sub-systems design. Unfortunately, it was not available during the study. One can also list the problems linked to units and exchange of binary code, which are not addressed by SMP2.

If the process timeline with the various tasks is represented on one axis, the SMP2 benefits area can be shown in Fig. 8.

It is also interesting to note that the process using SMP2 can be applied recursively at different levels of the system design. The same process can be applied to the design, integration and validation of a spacecraft sub-system.

The SMP2 standard is a valid approach for development. It allows focusing on engineering rather than on infrastructures. Sharing source code is not an issue but attention needs to be paid to the sharing of binary code.

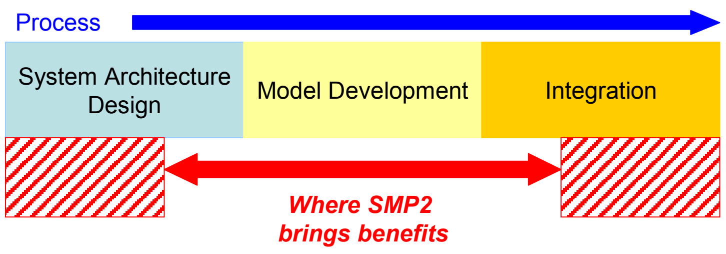


Fig. 8. SMP2 benefits in an industrial process

Regarding the SMP2 Tools

Concerning the tooling support, improvements are necessary. The SIMSAT4 used in the study was a pre-release version with many problems. It is now in a more stable state: the same simulation can be re-executed successfully on the new version. STOOD needs further improvements for SMP2 support and for the MMI. A GUI is obviously more than desired in simulation design activities (exists in STOOD but it needs to be further developed).

Possible Points for Future Studies

Finally, the VISMP2 study should be completed to address the following points:

- Design of complex OBC models and interfaces (SVF type simulator).
- Real-time tests (test beds or simulators with HW in the Loop).
- Integration of external data (e.g. Satellite Database).

Recommendations for Evolution

The VISMP2 study output has the following recommendations for the SMP2 evolution:

- Support of external data (like Satellite Database).
- Graphical notation for SMP2.
- Improvements for sub-assemblies support. For example, the ability to cross-reference items in different sub-assemblies to create model links between assemblies.
- Recommendation for simulation infrastructures to perform compatibility checks of units between model fields.