A Model-Based methodology to support the Space System Engineering (MBSSE)

S Mazzini, E Tronci, C Paccagnini, Xavier Olive

A Model-Based methodology to support the Space System Engineering (MBSSE)

S. Mazzini[1], E. Tronci[2], C. Paccagnini[3], X. Olive[4]

1: INTECS, Via Umberto Forti 5, Ospedaletto I-56121 Pisa, Italy

2: La Sapienza University of Roma, Via Salaria 113 I-00198 Roma Italy

3: Thales Alenia Space, Strada Antica di Collegno 253 I-10146 TORINO Italy

4: Thales Alenia Space, 100 Boulevard du Midi, BP 99 06156 CANNES LA BOCCA France

Abstract: This paper presents a model based methodology that relies on the sound basis of the most recent and widespread applicable system engineering standards and model based practices, The methodology has been defined to support domain specific space system engineering standards and practices and assessed through the application on industrial case studies. A complementary formal verification approach has also been experimented.

Keywords: model based, system engineering, formal verification, methodology, process.

1. Introduction

The use of modeling techniques at different stages of the development process is of crucial importance in order to successfully realize space complex systems.

The System and Software Functional Requirements Techniques (SSFRT) (ESA/ESTEC Contract N. 21188/08/NL/JD) was a study for the application of model based engineering technologies to support the space system-software co-engineering development processes, from mission level requirements to software implementation through model refinements. It aimed at making the software constraints present since the system analysis and avoiding any rupture during the development process.

The SSFRT study objective was to investigate how the system and software requirements processes can be modeled and interrelated with special emphasis during the early phases of a space system project, through the application of SysML and the integration of complementary and domain-specific modeling technologies.

SysML is considered a very promising system modeling language, its usage being currently encouraged both by INCOSE and NASA in their System Engineering Handbooks.

the ASSERT project constitutes a background for the SSFRT study, that intended to complement the ASSERT results at system level.

The study has resulted in the definition of the **M**odel-**B**ased methodology to support the **S**pace **S**ystem **E**ngineering (**MBSSE)** [1].

2. MBSSE Overview

The MBSSE methodology addresses the early phases of the system life cycle, in particular the ECSS Phases 0, A, and B, from the early definition of mission needs to the elaboration of a feasible, preliminary system definition.

It was conceived with reference to the technical processes and the principles defined in the ISO/IEC 15288, the major system engineering standard providing a common process framework that can be applied as a reference for any domain to cover the life cycle of complex computer based systems.

The MBSSE methodology is based upon a model-centric definition of the system using SysML and adopts, where feasible, the model based engineering approach to integrate complementary and domain specific modeling languages.

The MBSSE methodology deals with system requirements as first class citizens; they are captured, visualized and traced along the modeling process. The SysML model is the work product of the methodology, describing and justifying the requirements analysis in relation with the functional analysis and the modeling of the system structure and functional allocation.

Other languages and profiles may be integrated for system analysis (Matlab, Petri Nets), to apply discipline specific techniques (e.g. Simulink), or to perform system and software co-engineering (UML, MARTE, AADL, SOC, SCADE/Esterel, HRT-UML/RCM, SDL).

The MBSSE methodology is compliant with the following ECSS standards:

- the ECSS-E-00 [2], which sets the basic rules and overall principles to be applied to all space engineering activities during the performance of a space project [ECSS-E-00];
- the ECSS-M-10 [3], which defines the phases and the activities of space projects [ECSS-M-10]
- the ECSS-E-10 [4], which guides organizations involved in the development of systems for space applications by specifying the system engineering requirements in relation with the system engineering functions (*requirement engineering*, *analysis, design and configuration, verification, integration and control*) and the specific phases of space systems (0, A, B, C, D, E, F);
- The ECSS-E40 [5], which defines system and software co-engineering requirements related with software development;

The MBSSE allows you to deliver the required documents (MRD, RB, etc.) associated to the engineering activities, as specified in the ECSS-E-10, during the design process. Each activity and their related models are compliant with the mandatory documents required to support project review objectives as specified in ECSS-M-10 (MDR, PRR, SRR, PDR, etc.).

In addition the MBSSE methodology strongly relies on and integrates from the following bases:

- The ISO/IEC 15288:2008 System Engineering – System Life Cycle Processes [6] International Standard;
- The ISO/IEC TR 19760 System Engineering – A guide for the application of ISO/IEC 15288 (System life cycle processes) ([7]).
- The INCOSE Systems Engineering Handbook [8], that provides a large guideline for system engineering with the application of the ISO/IEC 15288 standard.

The approach for the definition of the methodology is:

- to support space system engineering according to the ECSS standards, with emphasis on the application of the E10 for the spacecraft segment on the early phases 0, A and B, aimed at achieving a feasible system specification,
- to integrate with the ISO/IEC 15288 technical processes as reference processes,
- to adopt a model-driven solution using the SysML language, from the mission requirement capture to the model of the system feasible specification.

Rationales are the following:

- SysML was conceived with reference to the overall system engineering processes and principles, as defined in the ISO/IEC 15288
- In particular SysML is able to cover the space system engineering processes, from high–level requirements to the architecture definition and verification.

The steps to achieve the definition of this methodology are the following:

- The ISO technical processes that are relevant to the context of the methodology were identified
- A mapping from the ECSS-E-10 functions to the relevant ISO technical processes was identified
- A correspondence between the ECSS phases 0, A and B to the ISO processes was identified
- The relevant ISO technical processes were initially tailored/integrated, in order to completely fit in this framework.

The following figure illustrates the overall description of the methodology resulting from the above described theoretical approach.

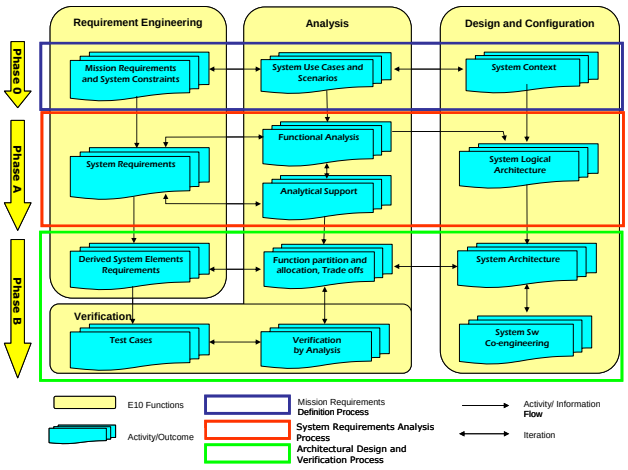

Figure 1 The MBSSE Methodology

Three main System Engineering Processes identified for the MBSSE methodology, are derived from the ISO processes with the purpose to specifically address the E10 functions and the space project initial phases:

- The *Mission Requirements Definition* Process, mainly addressing Phase 0 activities (derived from the ISO Stakeholder Requirements Definition Process),

- The *Requirements Analysis Process*, mainly addressing Phase A activities (derived from the ISO Requirement Analysis Process),
- The *Architectural Design and Verification Process*, mainly addressing Phase B activities (derived from the ISO Architectural Design Process).

The MBSSE processes support the Phases 0, A and B of the space projects, as defined in the ECSS standards.

They are conceived to be applied iteratively across the project phases in order to reach the specified milestones. At each iteration, outputs are evaluated, refinements are identified and trade-offs between different criteria or between alternatives are analyzed and discussed with the different stakeholders and system discipline specialists; this has the purpose to produce more refined and consolidated outputs that meet stakeholder expectations in a transparent and traceable way.

The activity flow, resulting from the iterative application of the processes, will lead to the preliminary definition and validation of the system model, as well as to the definition and consolidation of the associated requirements, from the identification of mission needs down to the physical description of the system elements.

The figure 2 shows the MBSSE process iterations across the ECSS project phases and reviews and minor and major deliveries for the reviews.

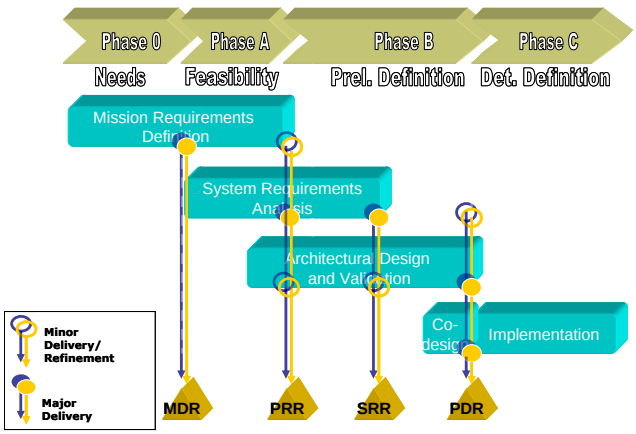

Figure 2 The MBSSE Processes and Iterations

Major deliveries represent consolidated outcomes that represent transition gates from a phase to the following one. Minor deliveries may represent initial or intermediate results that are preliminary to major outcomes, or refinements that may arise after major outcomes.

As showed in figure 1, each MBSSE process is composed by activities with the following peculiarities:

- Each process activity may be iterated several times, in order to achieve a satisfactory outcome at each project phase.
- Process activities may be exercised concurrently during each project phase.
- Process activities may be exercised, and equally valid and necessary, at all levels of decomposition within the space product.
- Process activities address lower levels with greater thoroughness as the project progresses.

The MBSSE processes, activities and related flows, that need to be carried out during each system phase, are defined by a tailoring of the ISO processes, activities, in order to integrate all the ECSS requirements and to be supported by means of SysML models.

The MBSSE methodology is that it extends at the system level the validity of the *separation of concerns* notion issued from the ASSERT project, by confirming such results at the software level.

Functions, activities and behaviors representing functional parts (function tree), are elaborated by the *Analysis Function* and modeled at system level separately from the non functional part, represented by the definition of the system structure, interfaces, boundaries and constraints (product tree), elaborated by the *Design and Configuration Function*. The two models are associated by allocation of functional parts of the function tree to the hardware/software elements of the product tree and the allocation of software components to hardware components.

This enables specification of the functional behavior of software components, but also, as a consequence of the allocation to the hardware, the inference of most of the non functional properties of the software components.

Today system engineers are using the function tree implicitly. They have no use to represent it explicitly.

The “function tree” need is coming from the new technologies which can be used on-board like TSP, multi-core, …, and is required to allow a flexible allocation into the hardware.

By instance, let consider a star tracker with a hardware component (CCD sensor) and an associated function (image processing). The function can allocated to the star tracker itself (i.e. we have a smart sensor) or on the central computer (i.e. the

star tracker is reduced to a CCD sensor). This trade-off can be done at system level, only if the function tree is available and contains the “image processing” function.

Finally the function tree representation at system level allows to remove a part of the implicit thing, reducing the design risk.

Figure 3 shows an high-level view of the system engineering model, partitioned into the collection of related sub-models, each representing the outputs of one of the main areas of the space system engineering functions.

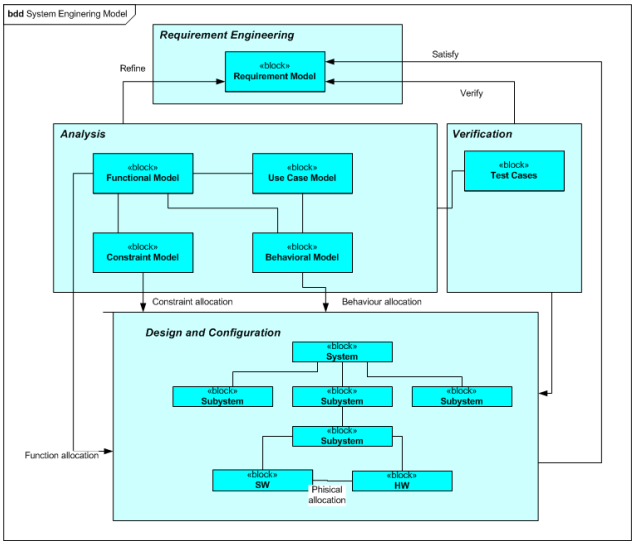

Figure 3 Model partitioning

The MBSSE methodology is based on the definition of a SSE Profile of the OMG SysML and UML, that defines a domain specific language for Space System Engineering taking into account modeling practices currently adopted in the space domain, as well as the ECSS standards.

3. The Case Study

The MBSSE approach has been applied by Thales Alenia Space on a real case study. The case study is related to the ExoMars mission, and covers only partially its perimeter. Both France and Italy were involved in the project. This opportunity has been used to highlight the fact that model can be exchanged. Italy has been in charge of phase 0 and A modeling. The model has then been sent to France, which has modeled the phase B part.

Phase 0 modeling has been focused on the definition of the mission requirements. Textual requirements have been modeled. Mainly table (and not graphical one) representation has been used for visualization (Figur3 4). System context has been defined by the use of SysML Block Definition Diagram (BDD). Data and/or control flows have been preliminary defined by the use *flow specification*. Finally use case and activity diagrams have been used to model the mission case and scenario.

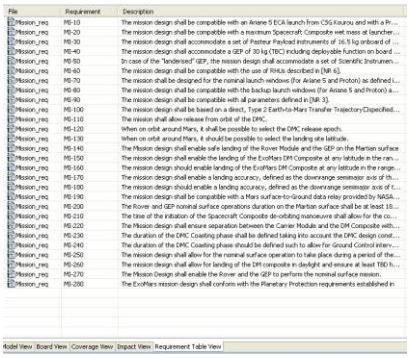

Figure 4 – Requirement table

Phase A deals with the system requirements refinement. Mission requirements have been refined by using classical SysML requirement management relationship. A functional architecture has been defined based on the system requirements set. A preliminary physical architecture, called system logical architecture, is used as reference to perform some trade-off and verification analysis. This architecture can be inherited from past project, or a new one from scratch. Traceability is ensured between both architectures and requirements set. In addition, they are both compliant with the previously defined flows during phase 0. Functional architecture is refined, by the addition of sub-function. In some particular functional chain, some skeletons for analytical support can be generated. By instance, for AOCS (Attitude and Orbit Control System), each function (AOCS mode, AOCS specific function) can be mapped to a Matlab/Simulink block. It allows you to have the same design in the functional architecture and in the analytical support model.

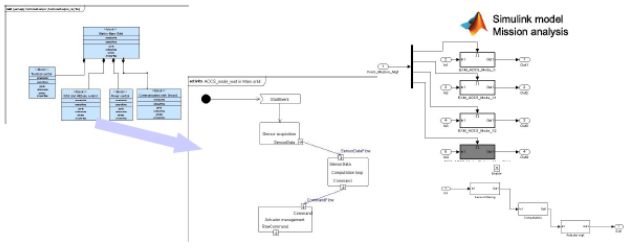

Figure 5 Functional architecture and analytical support

Phase B covers the consolidated definition of the system. The requirements are refined and grouped by logical subsystems. Requirements baseline is available for each functional subsystem. The

refinement takes into account the constraints coming from the functions allocation and trade-off analysis. The system architecture is consolidated. All the interfaces are fixed. The product tree is defined based on the system logical architecture and on physical units, supporting the allocation of functions. Data and control flows are refined and allocated to the defined interfaces.

Based on the current practices in Space domain, system-software co-engineering is only addressed in phase B. Due to paper length limitation, the complete activity can be reported. First the model has been transformed from SysML to UML modeling language. Based on the UML model and the interface definition, a LwCCM model has been derived to represent the software architecture. The non-functional (NF) properties (real time constraints, dependability, …) are derived from software requirements, and added by annotating the LwCCM model. **Figure** Figure 6 provides an overview of the used models and their interactions. The main result is the conservation of the separation of concerns, issued from ASSERT, at system and system/software levels.

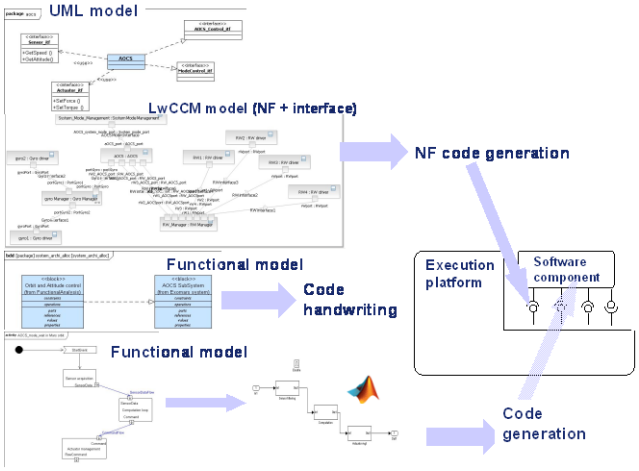

Figure 6 Separation of concerns

The application of MBSSE approach on a real case can be considered as successful from the methodological point of view. Nevertheless the tools are still not enough mature to edit and handle model. Some transversal functionalities like frozen a part of the model to provide it to a third party, or managing the model in a configuration management system.

4. System Level Formal Validation

The MBSSE methodology includes the application of model checking techniques for System Level Formal Validation.

System Level Formal Verification (SLFV) has the goal of showing that requirements for the subsystems a given system consists of are correct with respect to the (overall) system requirements. In other words, SLFV has the goal of guaranteeing that if each subsystem satisfies its own requirements then also the system built out of such subsystems will satisfy its requirements. Accordingly, SLFV can be an effective tool to support separation of concerns, since, by using SLFV, we can guarantee that the requirements for the subsystems are correct.

From the above we see that SLFV can be seen as a *Validation* activity (answering the question: are we building the right system?) for the subsystems and as a *Verification* activity (answering the question: are we building the system right?) for the overall system.

Accordingly, SLFV, together with testing and simulation, can support *System Level Functional Analysis*, more specifically, system level V&V. In this respect note that testing and simulation are geared towards showing presence of errors, thus they can only provide evidence for a negative answer to V&V questions. On the other hand, SLFV techniques (e.g. model checking [9]) are geared towards showing absence of errors, thus SLFV provides evidence for a positive answer to V&V questions.

The classes of system models handled by the above mentioned analysis techniques are also different. In fact, testing and simulation can handle quite detailed models as long as all inputs and parameters are defined. On the other hand, formal techniques can handle models with undefined parameters and inputs (e.g., modeling faults or disturbances) as long as such models have a moderate size. Thus, formal techniques can be used for a worst case analysis returning as output a worst case scenario (i.e. inputs and parameters) for the system under analysis.

The above considerations suggests using formal techniques as early as possible in the system design activities, namely: as soon as some system model (even qualitative) is available. In fact, this allows early detection of errors in system or subsystem specifications. For example, as for space software development, the above considerations suggest using formal techniques towards the end of phase A or at the beginning of phase B.

4.1. Supporting Proof Continuity with SLFV

In our framework we can use SysML to define the system structure as well as the interfaces between subsystems. This make such information widely available to system engineers. We can then use specialized languages to define subsystems behaviors or requirements.

Resting on the SysML system description, SLFV can support proof continuity by using an assume-guarantee verification approach to establish a formal

link between system level validation and subsystem verification. To this end we can proceed as follows.

First, we assume that subsystems (software or hardware) meet their requirements and verify (using model checking techniques) that the overall system meets its system level requirements. This verification activity indeed validates the subsystems requirements since it shows that subsystem requirements are correct (i.e. we are building the right subsystems).

Second, we guarantee that indeed the subsystems (software or hardware) meet their requirements. This is done by showing, for example by using model checking techniques, that the implementation of each subsystem satisfies the given specifications. Of course, depending on how the subsystem is implemented, a suitable model checker will be used.

Note that, as for model checking purposes, in the system level validation activity we map requirements into formal specifications and behavioral models into formal models (for a model checker). On the other hand, when using model checking tools to link subsystems formal verification to system level validation, we map model for subsystems into formal specifications and implementations for subsystems to formal models.

4.2. Model Checkers for SLFV

When modeling for SLFV we need to model the behavior of all subsystems the system under verification consists of. For example, in a control system this means that the controller as well as the controlled system (plant) will have to be modeled. In our context the controller is often software based whereas the plant is typically a physical system. This yields a dynamical system which state can undertake continuous (e.g. temperature, voltage, current in the controlled system) as well as discrete (e.g. mode of the controller policy) changes. Such systems are known in the literature as Hybrid Systems (HSs), for example see [10]. From the above considerations follows that when choosing a model checker to support SLFV we should consider focusing on model checkers for hybrid systems.

4.3. An Example: The Battery Manager System

In the SSFRT project SLFV has been applied to a case study proposed by Thales Alenia Space France: the Battery *Manager System* (BMS) shown in Figure 4. The CMurphi model checker [11, 12, 13] has been used to model BMS and to verify some given safety properties. Details are in [1].

We focus our SLFV analysis on the dependency of the BMS controller (BM in Figure 4) sampling time (T, in seconds, in Table 1) from the maximum voltage available from the solar cells (VS, in Volt, in Table 1).

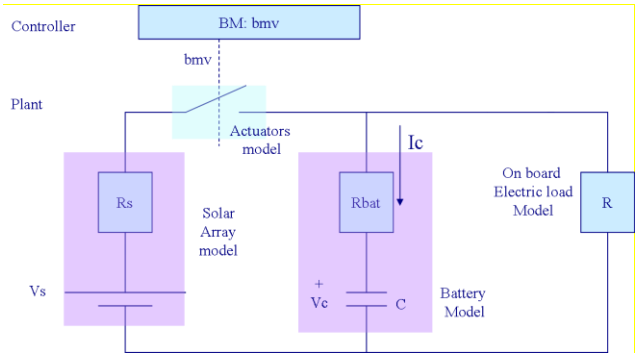

Figure 7 Battery Manager System (BMS)

Intuitively, BM policy should be correct as long as T is not too large with respect to the speed of system dynamics. Of course, the larger T, the larger BM reaction time and the easier the design of BM since it will have more time available for computing the command to be sent to the actuators.

Table 1 summarizes our findings. Column "CPU" gives the CPU time (in seconds) to complete the verification task using CMurphi. Column "SLFV Outcome" summarizes CMurphi output.

VS	T	CPU	SLFV Outcome
150	9	1216.16	No error found
150	10	117.47	No error found
150	11	24.72	Safety Violation found
200	9	1259.59	No error found
200	10	47.19	Safety Violation found
200	11	25.01	Safety Violation found

Table 1: Experimental results for BMS formal verification using CMurphi on a 2GHz Dual Core Linux Pentium PC with 2GB of RAM.

From Table 1 we see that the larger the voltage from the solar cell the smaller should be the controller sampling time. Thus, our SLFV activity effectively shows how software requirements depend on physical parameters, namely, the max solar cell voltage.

4. Lessons Learned

This section provides a description of the 3 main identified weak points and the 3 main strong points.

First weak point is related to the current Space engineering practice : The physical architecture appears too late. In the MBSSE methodology, the physical architecture is defined in phase B. Current habits are to have a first physical architecture in phase A. This point is not a blocking one but requires to change the way of designing and to make system engineers understand the commonality between a logical system architecture and a

preliminary physical architecture. Typically today the design approach used in CDF in more a bottom-up one (instead of descending one in MBSSE). This point can be considered as a positive for the methodology itself but is weak point in the frame of its adoption.

Second weak point is about SysML. It is a generic defined modeling language for system engineers across all the application domains (Space, Automotive, Nuclear, Airplane, …). In this sense it provides generic features which need to be specialized with regard to the Space domain. For instance, a block model element can represent a product, a function, a part, … During the study, an appropriate profile called SSE has been defined. We recommend to use some domain specific profile or language.

Finally the third weak point is about the maturity of (open-source) tools supporting SysML, so that it is still not possible to recommend a tool for specific industrial use. Main issues are about the transversal functionalities (sharing, versioning, splitting, …) and the way to deal with views, to enable the management of big-sized model.

One of the main contribution of the MBSSE methodology is to maintain the separation of concerns from systems engineering to software engineering. It is present during all phases 0, A and B. On one hand, functions, activities and behaviors representing the functional part are designed in an analysis activity; on the other hand, the non functional part is represented by the definition of system interfaces and boundaries, and then the product tree. The association between both is performed during phase B by allocating the functional part (function tree) to the non functional one (product tree).

A second improvement is about the traceability. SysML enables full traceability from phase 0 to phase B. With a restricted number of relations (mainly allocation, refine, derive), all the model elements can be linked altogether. Traceability is expressed from/to the requirements model element and can be extended to the other model elements (like software element / component).

The third main enhancement is to keep the separation between system and software engineering. System - software co-engineering activity is used to go from one world to the other one. For this activity, the system provides as inputs a set of functions/activities to develop (functional part), and a set of non functional elements. Allocation relations can be used to map system block to software components. Mapping can be one-one, n-one or one-n. The allocation relation introduces flexibility between system and software, and permits to have 2 different but compliant points of view of a software based system.

In complement to this specific advantages, MBSSE offers the classical ones issued from MDE approach: reuse of models, complexity coping, … .

5. Conclusion

There is a need for a common modeling language to improve communication and cooperation among the different space system domain disciplines, in order that they can work in an integrated way, and to realize seamless integration among the different space development phases.

The SysML language can provide an harmonization and integration models for the involved disciplines across the whole system life cycle, even if there is the need for defining a profile (such as SSE) or DSL to specialize SysML to model the entities that are handled by the Space system engineers.

Properties and behavior of system models shall be verified in the model instead of being just tested after development, in particular they can be formally verified by application of model checking techniques.

The use of models to support the system requirements engineering process was aimed at improving the system requirements allocation process towards the software requirements engineering process.

On the other side, design constraints and system requirements refinement imposed by software on the system functions realization can be easily taken into account if both processes are managed and supported by modeling techniques:

- System requirements may be fully traced to the software requirements.
- Mappings from the system modeling language to the software engineering modeling languages may be defined, according to the ASSERT approach of the separation of concerns.

6. Acknowledgement

We are grateful to the ESA ESTEC Project Officer Andreas Jung and Head of the TEC-SWE Division Jean-Loup Terraillon for the very useful discussions we had within the framework of the SSFRT project.

7. References

[1] S. Mazzini (ed.), S. Puri, X. Olive, C. Paccagnini, E. Tronci: "*Guidelines for Model Based Space System Engineering* ", SSFRT Report, 2009.

[2] ECSS-S-ST-00C, ECSS System, Description, Implementation and General Requirements, 31 July 2008.

[3] ECSS-M-ST-10C, Space project management, Project planning and implementation, 31 July 2008.

[4] ECSS-E-ST-10C, Space Engineering, System Engineering General Requirements, 6 March 2009.

[5] ECSS-E-ST-40C, Space Engineering, Software, 15 March 2008.

[6] ISO/IEC 15288, System engineering, System life cycle processes, ISO/IEC 15288:2008(E) IEEE Std 15288-2008, 2008.

[7] ISO/IEC TR 19760, System Engineering – A guide for the application of ISO/IEC 15288 (System life cycle processes), 2003.

[8] Haskins, Cecilia (ed.), INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, v. 3.1, INCOSE-TP-2003-002-03.1, International Council on Systems Engineering, August 2007.

[9] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, Cambridge, Mass., MIT Press, 1999.

[11] R. Alur, T. A. Henzinger, and Pei-Hsin Ho, Automatic symbolic verification of embedded systems, IEEE Trans. on Software Engineering 22:181-201, 1996.

[12] G. Della Penna, B. Intrigila, I. Melatti, M. Minichino, E. Ciancamerla, A. Parisse, E. Tronci, and M. Zilli. Automatic verification of a turbogas control system with the murphi verifier. Hybrid Systems: Computation and Control, LNCS, Springer, 2003

[13] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, M. Zilli: Exploiting transition locality in automatic verification of finite-state concurrent systems, International Journal on Software Tools for Technology (STTT) 6(4): 320-341 (2004)

[14] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, M. Zilli: Finite Horizon Analysis of Markov Chains with the Murphi Verifier, International Journal on Software Tools for Technology (STTT), Vol 8, N. 4-5, p. 397-410, Aug.