A Model-Based Systems Engineering Approach to the Design of a Spacecraft Subsystem

Rodrigo Jorge Silva Ramos
rodrigo.j.s.ramos@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

December 2021

Abstract

The systems engineering (SE) discipline has been extensively adopted in the aerospace industry. However, traditional SE approaches, which are centred on the development and control of written documents, are becoming too expensive. Model-based systems engineering (MBSE) aims to mitigate this issue by introducing modelling and information centralization into the traditional SE practices. MBSE has been applied successfully in conceptual design processes, but its application to the design of a spacecraft subsystem is yet to be fully demonstrated. In this work, a methodology based on the OOSEM is proposed for the design of a spacecraft subsystem, which includes a framework for integrating MBSE with simulation through a model transformation approach. This methodology is demonstrated by being applied to the design of an Attitude Determination and Control System (ADCS). Some benefits of MBSE are verified, and we conclude that an MBSE approach can be applied at the subsystem-level, while properly integrating systems engineering with other specialized domains of knowledge. We further demonstrate that model transformation approaches are capable of successfully generating fully functioning simulators from SysML specification.

Keywords: MBSE, SysML, Model Transformation, OOSEM, MATLAB/Simulink, ADCS

1. Introduction

1.1. Motivation and Objectives

The aerospace industry has historically been among the main proponents of Systems Engineering (SE), with some crediting this discipline as a main driver of success in the development of complex systems [5, 22]. As systems become more distributed and interconnected, and as their development becomes more de-centralized, the traditional mechanisms through which SE ensured consistency – written documents – are becoming increasingly inadequate [7]. Model-based systems engineering (MBSE), the application of systems modeling "to support analysis, specification, design, and verification of the system being developed" [8], can help mitigate this issue. With MBSE, information is centralized in a system model which represents a single source of truth, ensuring consistency and traceability and leading to improved complexity management.

In the last 15 years, the interest of researchers in MBSE has increased over time [13], but some obstacles still have to be overcome for MBSE to be fully adopted in the industry. Some research has been done showing that MBSE can successfully be applied in support of system-level conceptual design processes [14, 23]. However, when applied to the design of subsystems, MBSE faces the challenge of integrating the systems engineering discipline with specialized domains of knowledge. In specific, the system model created with MBSE must be integrated with discipline-specific engineering models and simulation.

In this work, a methodology based on the OOSEM is proposed for the design of a spacecraft subsystem, which includes a framework for integrating MBSE with simulation through a model transformation approach. The methodology is demonstrated by being applied to the design of an Attitude Determination and Control System (ADCS) of a small satellite, where simulation is used to evaluate subsystem performance and compare alternative designs.

We apply MBSE in support of a design process of a spacecraft subsystem, inside the scope of a spacecraft design process. This constitutes a case study that is not currently available in literature. In addition, the application of model transformation approaches to integrate MBSE with simulation is rare in literature, and the validity of this type of approach in the context of complex dynamic behaviour simulation is still to be assessed.

The remainder of this work, starts with a review of literature followed by a discussion of the most relevant concepts to the development of this work. Then, we discuss the proposed methodology, including the developed model transformation framework, after which we demonstrate it as applied to the design of an ADCS. In the end, performance simulation results are presented.

1.2. Literature Review

Some works have been developed concerning the application of MBSE to the design of spacecraft subsystems. The derivation of subsystem requirements from customer needs and requirement verification using simulation have been performed for an ADCS [11]. A development process for AOCS software using SysML has been developed and integrated with code generation [21]. Also, the design of an ADCS has been developed with the Vitech MBSE methodology [12], and a satellite communication system architecting framework has been developed with the OOSEM methodology [10]. There is a lack of case studies that present an MBSE approach applied to the design process of a subsystem inside the context of a complete spacecraft design.

There are two common approaches to integrate MBSE with simulation: co-simulation and model transformation [18]. Co-simulation is the more mature of the two approaches. It has been used to enable the reuse of a previously developed ADCS simulator [11]. A co-simulation approach was also developed to integrate SysML with MATLAB scripts and other more complex simulation workflows which integrate multiple simulation tools [15]. These approaches are in general not scalable for more complex projects [18].

Model transformation approaches are much less mature, mostly because of the lack of standard maturity and tool support. This type of approach has been applied to the simulation of a simple electrical circuit, using the SysML2Simulink standard [2], and to the simulation of the discrete/continuous behaviour of an inverted pendulum system, using a method based on triple graph grammar [1]. These and other examples available in literature concern in general very simple systems.

2. Background

In the context of MBSE, a methodology is defined as a set of related processes, methods, and tools used to support the systems engineering effort [6]. A review of some of the more notable MBSE methodologies has been done in [6], and from these, the Object-Oriented Systems Engineering Method (OOSEM) [8] was selected to be used in this work, because: (i) it is well documented; (ii) there is an extensive example of its application in the design of a space system [9]; and (iii) many of the reviewed works use this methodology [15, 23, 10].

The OOSEM is a top-down, scenario-driven process, which combines modelling techniques and a solid systems engineering foundation with object-oriented concepts [8]. It uses the Systems Modelling Language (SysML) [20], a general-purpose graphical modelling language, to "support analysis, specification, design, and verification of systems" [8].

The MBSE modelling tool used in this work is the Cameo Systems Modeler (CSM) from NoMagic, and this tool is integrated with MATLAB/Simulink using a model transformation approach. This type of approach consists of the automatic transformation of the systems models into executable simulation models. With it, a simulation model can be specified almost completely inside the model repository and the mathematical modelling and simulation can be performed inside the context of a simulation environment.

This approach, as opposed to co-simulation, was selected because it ensures a better centralization of information in the system model and because the development of mathematical models inside the context of a simulation environment is needed to enable model reuse. Also, this is the type of tool normally used by domain specialist engineers.

The methodology proposed in this work concerns the preliminary design of a spacecraft subsystem. To support this design process, a model transformation framework is developed to enable the generation of a Functional Engineering Simulator (FES) [4] from SysML specification. This can be used to derive values for a set of Technical Performance Measures (TPM), i.e., quantitative attributes of system elements. The framework leverages the SysPhS standard [19].

3. Approach

3.1. Methodology Adaptation

In this work, we propose an MBSE methodology tailored for the preliminary design of a spacecraft subsystem, which results from the adaptation and extension of the OOSEM. In specific, the OOSEM is modified with the purpose of adapting it to a subsystem design process. It is also extended to include the integration of simulation into this design process. The model developed in this work reuses the organization of the model developed in [9], as well as some of its elements.

Because the spacecraft subsystem is designed in the context of a complete spacecraft design, the application of some activities is a pre-condition to this work. These include the analysis of stakeholder needs, the OOSEM process applied at the system-level, and the complete analysis of subsystem requirements, except for the subsystem state machine behaviour definition. The artifacts that results from the application of these activities are generated using the model in [9] as a reference.

The OOSEM is modified to include the definition of three levels of component abstraction, instead of the two levels defined in [8] (the same approach is followed in [14]). These three layers are realized by the definition of three different subsystem architectures: functional, in which components are defined in terms of their function; logical, in which these are defined as a particular implementation of the available technology; and physical, in which the specific component design is completely specified (e.g., a particular procured COTS option). Additionally, the analysis activities and evaluation of multiple candidate architectures are only performed at the physical abstraction layer, where component characteristics can be easily determined.

3.2. Methodology Definition

The definition of the functional and logical architectures follows three steps:

1. Defining the subsystem decomposition.
2. Defining interactions between components to realize each subsystem activity.
3. Defining the subsystem Internal Body Definition (**ibd**) diagram, which specifies its internal structure.

At the logical level, software and hardware architectures are also defined, which are later integrated with those defined for each of the other subsystems. Also at this level, the critical properties of the components are identified and modelled as *value properties* in SysML. These properties and the application of the listed activities both support the final specification of the logical component requirements.

At the physical layer, component options are specified through the definition of values for each of these critical properties, and through the redefinition of component multiplicities. Different architecture alternatives are compared, and as a result, the preferred architecture is selected.

The management of requirement traceability is performed alongside the definition of these architectures. This activity includes: (i) the capture of text-based requirements in the model; (ii) the definition of relationships between requirements and other elements; and (iii) the continuous management of requirement updates and analysis of traceability gaps. The specification of the subsystem requirements, which guide the architecture definition process mentioned above, is part of this activity.

The verification of some of these requirements and the comparison of subsystem designs must be done using system performance analysis. Since the system in question has a complex dynamic behavior, this analysis must be performed using simulation. In the context of an MBSE approach, performing simulation consists of executing the specification contained in the system model. In this work, this is implemented using a model transformation approach that transforms this specification into a model specified with executable semantics, which can be used for simulation. In this case, the simulation model is expressed in Simulink.

3.3. Integrating MBSE with Simulation

The developed model transformation framework aims to generate a Functional Engineering Simulator from the specification of a single *analysis context*. The *analysis context* is composed of the system, the environment, and a *Monitoring and Control* block, which consists of scenario definition and post-processing functions [4]. Before the transformation is performed, and the simulator is generated, the SysML specification must be developed following these steps:

1. Modeling the components of the system and environment in SysML and Simulink.
2. Specifying the system and environment in SysML.
3. Specifying the *analysis context* and the *Monitoring and Control* block.
4. Defining values for parameters of the system, the environment, and the simulation.
5. Creating tables in SysML outlining key aspects and synchronising them with Excel spreadsheets.

The process of performing engineering analysis consists of applying the model transformation process, which transforms the specification obtained through the steps listed above into a simulation model. This includes the use of the *Simulink Export* function of CSM. This function leverages the SysPhS standard to generate a simulation model from a SysML block. To extend this function's capabilities, a MATLAB program was developed. After the two are executed, the final generated Simulink model is ready to be used.

The *Simulink Export* is capable of generating the "skeleton" of the simulation model, by exporting the *analysis context* block. This "skeleton" includes the decomposition of the *analysis context*" into blocks, the external interface of each of the composing blocks, and the connections between composing blocks. The lowest-level elements, named components, are exported as black boxes, since this is how they are specified in SysML.

These components that are defined in SysML are also modelled mathematically in Simulink, because SysML semantics are not adequate for modelling behaviour of algebraic nature. This is also appropriate since domain specialist engineers already use this type of tool for this activity (i.e., mathematical modelling).

The Simulink mathematical models are used to replace the exported SysML black box components in the simulation model. Therefore, the black box representations in SysML and Simulink must be consistent. The ports contained on the SysML block must match the inputs and outputs of the Simulink block, and the parameters used in the mathematical model must be identified in SysML as *value properties* of the component. In SysML, the mathematical model that represents the behaviour of each component must be identified.

The behaviour of the system and all its composing elements should be decomposed down to the component-level, so that it may be modelled mathematically in Simulink. However, the behaviour of complex systems in general cannot be completely decomposed into behaviour of components. To address this issue, this framework can also transform SysML state machines into Stateflow blocks defined in Simulink. This is accomplished with the MATLAB program that was developed.

In SysML, state machines specify the behaviour of elements in terms of the states in which they may be. It is common for system elements to behave differently depending on the state of other elements (e.g., the operation of an attitude filter may depend on the active ADCS mode). For this reason, it is essential for state machines to be included in the final simulation model.

The active state of each state machine during simulation is globally accessible in the model. This means that mathematical models can use these active states in their definition, as they would in a regular simulation context.

Internal simulation parameters are specified in SysML as *value properties* of blocks. This is important to ensure the centralization of information into a single source of truth. The values of the parameters are transferred from SysML specification into the simulation model by the MATLAB program.

As mentioned above, in some cases, the SysML semantics are not adequate fro the representation of concepts that are algebraic in nature. To address this issue, the SysPhS standard provides the capability of defining components inside SysML that correspond to Simulink native block (i.e., it enables the import of Simulink blocks into SysML). When the model transformation process occurs, these imported block will be automatically converted into the original Simulink native blocks.

The *Monitoring and Control* block is essential in the simulation process. It is composed of blocks that compute error indices, from time-dependent simulation parameters, which are later used to derive TPMs. The TPMs are, in turn, used for performance requirement verification. Additionally, the *Monitoring and Control* block is also responsible for defining the simulation scenario that is executed. With this framework, a simulation scenario is defined by a set of SysML *signal events*, which are transformed into Stateflow *messages*. These *signal events* trigger transitions in the state machines, which all together define the simulation scenario implemented. This way, the simulation scenario definition is also centralized in the system model.

Finally, the framework also enables the definition of general simulation parameters in SysML (e.g., the Simulink model properties *Stop Time*, *Start Time*, and *Step Size*). The Simulink properties *StartFCN* and *StopFCN* can also be specified in SysML. These enable the definition of the functions required for the setup of the mathematical models, and for the post-processing of error indices and generation of TPMs, respectively.

4. Methodology Demonstration

4.1. Design Context

The methodology proposed in this work is demonstrated by being applied in the design of an ADCS of a 1U CubeSat. The mission objective is the observation of Earth, and in this context the ADCS must be able to orient a payload with the nadir.

The purpose of this demonstration is to assess the validity of the proposed methodology. The ISTSat-1 CubeSat mission [17] is used in this work as a basis for defining realistic characteristics of the system and mission. The architecture of the mission is shown in Fig. 1.

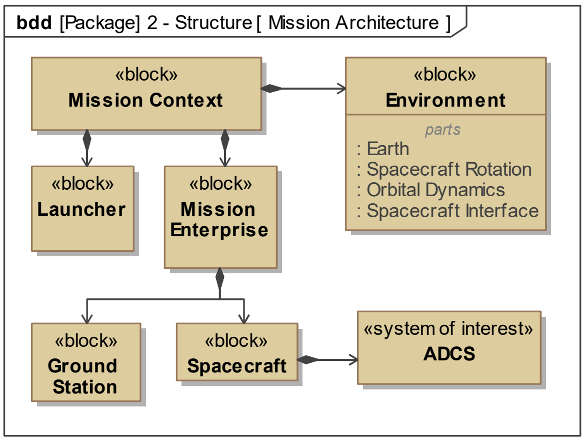

Figure 1: Mission architecture.

The mission enterprise comprises the systems which are the solution of the mission objectives, and whose definition is under the control of the development team (in this casse, the launcher is controlled by a third party). The focus of this work, at the system-level, is on the design of the spacecraft. However, the system-of-interest is the ADCS, which is considered a subsystem of the spacecraft. The mission context integrates the mission enterprise with other systems relevant to the mission and the environment.

Further context is provided for the design process with the definition of: (i) the system behaviour, using a state machine in SysML; (ii) the interactions of the ADCS with other subsystems and the environment; and (iii) the black box specification of the subsystem, which includes the definition of a list of requirements that the ADCS must satisfy.

These requirements are the major driver of the subsystem design process, and they are divided into operational, functional, and performance requirements, and design constraints. The design constraints are used in this work to limit the scope of the design process to an adequate degree for the application of the MBSE methodology.

4.2. Subsystem State Machine Definition

The subsystem design process starts with the definition of its behaviour using a SysML state machine. The ADCS has three modes of operation: (i) *normal pointing*, representing the regular operation of the subsystem; (ii) *safe pointing*, representing a safe state of the subsystem; and (iii) *detumbling*, used to reduce the angular rate of the spacecraft in the start of the mission or in failure situations.

Fig. 2 shows a view of this state machine which contains all the autonomous mode transitions originated in the *detumbling* mode. It shows how a state machine is modelled and viewed in SysML. As mentioned in Section 3, this state machine can be transformed into a Stateflow block when the simulation model is generated.

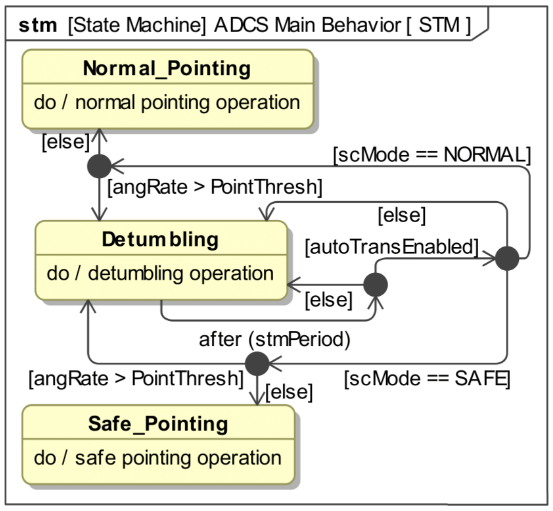

Figure 2: Representation of the autonomous transitions originated in the *detumbling* mode.

In this state machine, possible autonomous transitions are evaluated periodically. These only occur if they are the flag *autoTransEnabled* is true. When the angular rate is too high, the *detumbling* mode is activated, otherwise one of the pointing modes is selected according to the active spacecraft mode. The spacecraft has a *normal* mode and a *safe* mode, in which *normal pointing* and *safe pointing* may be activated, respectively. The spacecraft state machine is also exported with the model transformation approach, to be included in the simulation model. Additionally, ADCS mode transitions may also be forced by an operator, when required.

4.3. Subsystem Architecture Definition

The functional and logical levels of the subsystem architecture are defined in a similar manner. The existence of the two levels of abstraction helps the systems engineers to completely identify and decompose functions, and to avoid making early design decisions that inadvertently constrain the design space. During the design definition, the functional components are allocated to the logical ones, introducing traceability into the design. Fig. 3 shows an example of this process as applied to the hardware components of the ADCS.

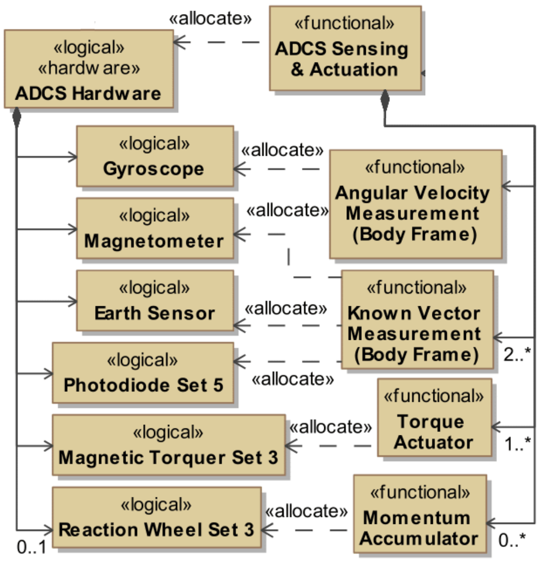

Figure 3: Example of allocation of functional components to logical components.

Here, we will focus more on the logical level of the architecture. The ADCS is composed of a gyroscope, a magnetometer, one photodiode installed in each face of the CubeSat (except for the face pointing to the nadir), and an Earth sensor (installed in the face pointing to the nadir). The attitude determination, performed with an attitude filter, combines the measurements of the gyroscope with two vector measurements. In *safe pointing* mode, the magnetometer and photodiodes ared used (the latter for estimating the sun vector). In *normal pointing* mode, the Earth sensor is used in the place of the photodiodes.

In terms of actuators, the ADCS is composed of three magnetic torquers and may or may not include three reaction wheels (the decision is made later, using simulation to compare alternative designs). In *safe pointing* mode, only magnetic torquers are used. The ADCS also includes on-board estimation of the spacecraft position. The attitude controller algorithms for the pointing modes are based on PD control techniques.

As discussed in Section 3, after the subsystem is decomposed into components, interactions between components are modelled to realized subsystem activities. In Fig. 4, an example of one of these interactions is presented, concerning the operation of the ADCS in the *detumbling* mode, at the logical level of abstraction.

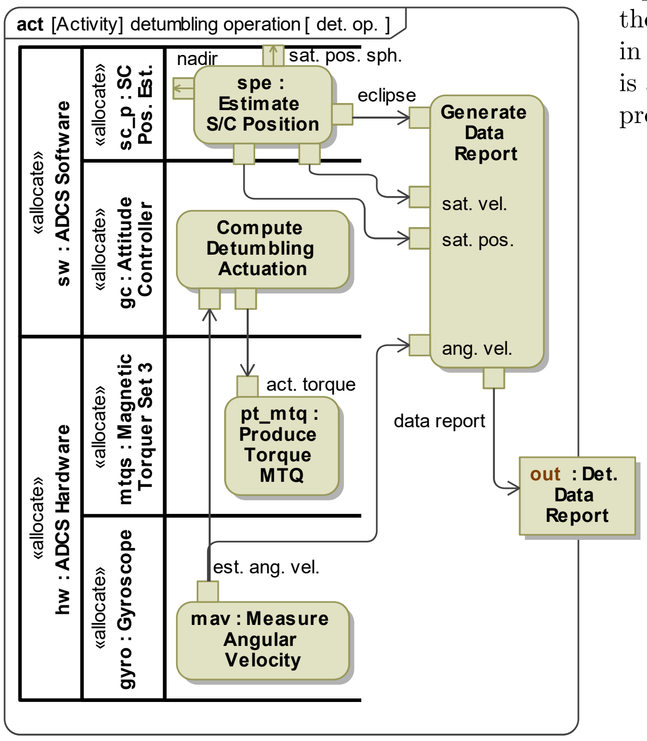
act [Activity] detumbling operation [det. op.]
«allocate» sw : ADCS Software
«allocate» hw : ADCS Hardware
«allocate» sc_p : SC Pos. Est.
«allocate» gc : Attitude Controller
«allocate» mtqs : Magnetic Torquer Set 3
«allocate» gyro : Gyroscope
nadir
sat. pos. sph.
spe : Estimate S/C Position
eclipse
Generate Data Report
sat. vel.
sat. pos.
ang. vel.
Compute Detumbling Actuation
act. torque
pt_mtq : Produce Torque MTQ
data report
out : Det. Data Report
est. ang. vel.
mav : Measure Angular Velocity

Figure 4: Interaction between components realizing the *detumbling* operation.

Actions, represented by rectangles with rounded edges, are attributted to components of the subsystem with the use of *swimlanes* (the table-like structures shown in Fig. 4). *Token* exchanges are modelled to represent the exchange of signals between the different components. In this case, the *detumbling* operation consists of a measurement of angular velocity being processed by an algorithm that computes the detumbling actuation, which is produced by a magnetic torquer. In parallel, the spacecraft position is estimated with numerical integration, and a data report is generated. This data report is defined as an output, as it is sent to the avionics subsystem.

The specification of the internal structure of the subsystem follows, which is based on the specified interactions (such as the one in Fig. 4), and on the defined subsystem external interface. At the logical level the internal structure of the subsystem is also divided into hardware and software blocks, which represent the hardware and software architectures of the subsystem. These are integrated with those of the other subsystems.

In Fig. 5, part of the internal structure of the ADCS is represented, concerning its software architecture. Each of the *token exchanges* modelled in the *activities* like the one shown in Fig. 4 is attributed to one connector in the **idb** diagram represented in Fig. 5. The internal structure of blocks, represented in this type of diagram, is replicated in the simulation model that will be generated later in the implementation of this design process. This is accomplished with the model transformation approach described in the Section 3.

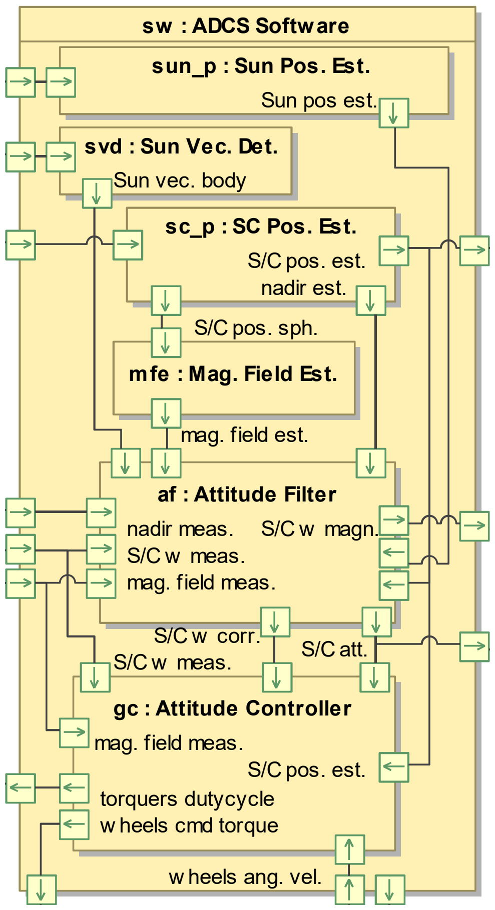

Figure 5: Internal structure of the ADCS software architecture.

As mentioned in Section 3, at the physical level components are specified with the definition of values for the components' attributes. Not all attributes are quantitative, since some of them represent methods and models used in the definition of the component.

For each hardware component, a COTS option was selected and the component attributes were defined according to the corresponding datasheets.

In terms of software, the Explicit Complementary Filter developed in [16] is used for attitude estimation. Attitude control is based on PD control techniques, except for the *detumbling*, in which a desired torque is computed as proportional to the angular rate and opposite to the angular velocity vector, in terms of direction. The spacecraft and Earth orbits are numerically integrated (from the latter is derived the Sun vector in the inertial frame). The magnetic field vector is computed on-board with the WMM 2020 model [3], and a Sun vector determination algorithm is used to convert photodiode readings into a Sun vector estimate.

Only two candidate physical architecture are synthesized in this work. These consist of designs with and without reaction wheels. They are compared in simulation, and requirement satisfaction is verified for each.

4.4. Management of Requirement Traceability

In the context of this activity, relationships are defined in the system model between each subsystem requirement and the elements that satisfy and verify that requirement. Requirements may be satisfied by multiple types of elements, such as *value properties*, *part properties*, or subsystem *operations*.

Some of the subsystem requirements are verified with simulation (e.g., the detumbling duration). All performance requirements are verified in this way (this concern the pointing error in *normal pointing* and *safe pointing* modes, the attitude knowledge error, and the spacecraft position estimation error).

4.5. Evaluation of Alternatives

Before simulation can be performed, the environment model must be specified. Its internal structure is specified in a similar manner to what was shown in Fig. 5. In Fig. 6, it is shown how the ADCS interacts with the environment. The interactions concern sensor measurements, actuation torques, and the spacecraft and Earth orbit updates required for the numerical integrations.

The environment is decomposed into four part properties: (i) *Orbital Dynamics*, which includes the orbits of the spacecraft and the Earth; (ii) *Earth*, which includes the Earth magnetic field and albedo; (iii) *Spacecraft Rotation*, which contains the rotation dynamics and torque disturbances; and (iv) *Spacecraft Interface*, which outputs the exact quantities that are measured by the sensors.

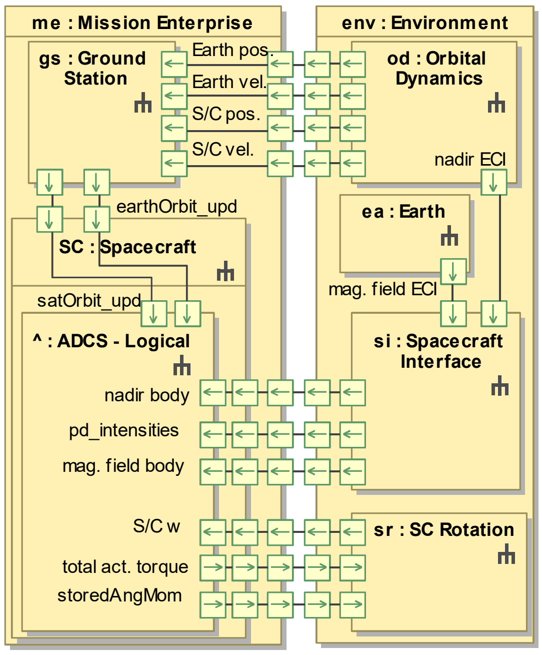

Figure 6: External context of the ADCS (simplified version).

Following the process outlined in five steps in Section 3, the specification of the *analysis context* and the *Monitoring and Control* block follows. Fig 7 shows the internal structure of the *analysis context* (it element can be seen as represented by the diagram frame). Some time-dependent parameters are sent from the mission context, where are they computed in the simulation model, and these are used by the internal components of the *Monitoring and Control* block to compute error indices. Each of these indices corresponds to one performance requirement. The obtained instantaneous values of these indices are processed after the simulation to obtain the value of each TPM.

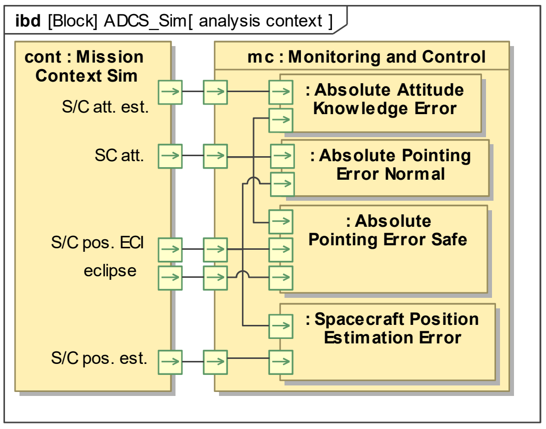
ibd [Block] ADCS_Sim[analysis context]
cont : Mission Context Sim
S/C att. est.
SC att.
S/C pos. ECI
eclipse
S/C pos. est.
mc : Monitoring and Control
: Absolute Attitude Knowledge Error
: Absolute Pointing Error Normal
: Absolute Pointing Error Safe
: Spacecraft Position Estimation Error

Figure 7: Internal structure of the *analysis context*.

Both candidate architectures defined above are simulated in the same scenario. This scenario starts with launcher separation, after which the detumbling is executed and the detumbling duration is evaluated. After the detumbling is done, the ADCS transitions to *safe pointing* mode, and it has two orbital periods to converge its stabilization, after which the performance of the ADCS in this mode is evaluated (regarding both pointing error and attitude knowledge error). This evaluation is performed during 10 orbital periods. The same process applies for the *normal pointing* mode, which is activated after the performance in *safe pointing* mode is evaluated. Each orbital period corresponds to 5668 seconds and the simulated scenario has a duration of 150000 seconds (around 1.7 days).

The scenario defined above is specified in SysML with the definition of *signal events* sent by the *Monitoring and Control* block. These *signal events* and the instants in which they are sent are specified with a SysML *interaction*.

The analysis of the simulation results is done here only regarding those that concern the *normal pointing* mode. In specific we consider the absolute pointing performance (APP) and the absolute attitude knowledge performance (AAKP). Both of these are TPMs of the ADCS. The values obtained for each of the architectures are shown in Table 1 (the architecture with reaction wheels will be named *A1*, and the other *A2*).

Parameter	**Value**	**Max.**
A1: APP (deg)	0.50	1
A2: APP (deg)	0.96	1
A1: AAKP (deg)	1.81	2.5
A2: AAKP (deg)	2.74	2.5

Table 1: Summary of the simulation results. The parameter values obtained with simulation (left) are compared to maximum values imposed by requirements (right).

We can conclude that both designs satisfy the required APP, even though the architecture *A1* shows a considerably better performance, as expected. The same does not apply to the required AAKP which is not satisfied by the architecture *A2*. This means that the magnitude of the pointing error has an influence on the magnitude of the attitude knowledge error, since the attitude filters are identical in each architecture. In this case, a trade-off between requirements may be done. While the architecture *A2* does not satisfy the AAKP requirement by a short margin, the architecture *A1* does not satisfy the defined mass requirement by a large margin. In this case we may update one of the requirements or consider different component configurations and designs.

With this approach, specialist engineers may also analyse any simulation result they deem relevant. In Fig. 8 the absolute pointing error is plotted against time, in the interval of 10 orbital periods in which *normal pointing* performance is evaluated.

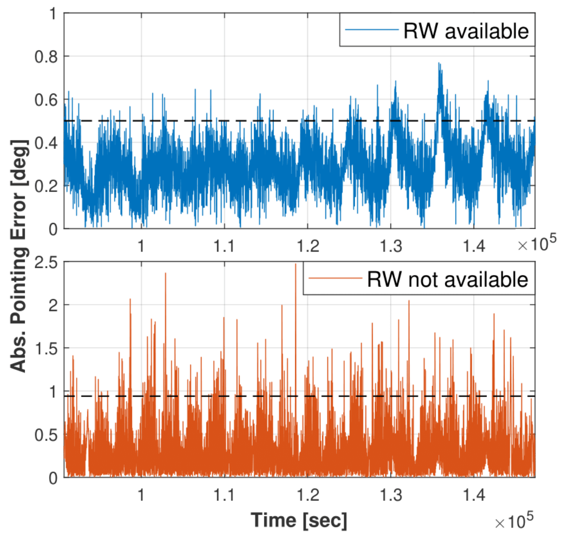

Figure 8: Absolute pointing error in *normal pointing*. The dashed line represents the TPM value.

The architecture *A1* (plotted above) has an absolute pointing error that never passes the limit of 0.8 degrees, because the reaction wheels can produce high torques in any direction. Contrarily, the other architecture (plotted below) shows spikes of pointing error up to a maximum of 2.5 degrees, which result from the fact that magnetic torquers can only produce torques perpendicular to the magnetic field, and also from the increased attitude estimation error.

In Fig. 9 the absolute attitude knowledge error is plotted against time, for both architectures (in the same interval as above). We can observe that the associated TPM value (represented by a dashed line) is increased due to the presence of error spikes, which occur when the nadir and the magnetic field vector are close to aligned. In fact, the estimation error always accumulates around the nadir, which means these error spikes have a very low influence on the absolute pointing error.

4.6. Discussion of Results

Some of the benefits of MBSE mentioned in literature were perceived in this work. The improved ability to manage complexity is observed in the breakdown of the subsystem behaviour. Also, the automatic propagation of changes through the model and the evaluation of model correctness helped to improve the quality of the specification and to reduce the time required to produce it.

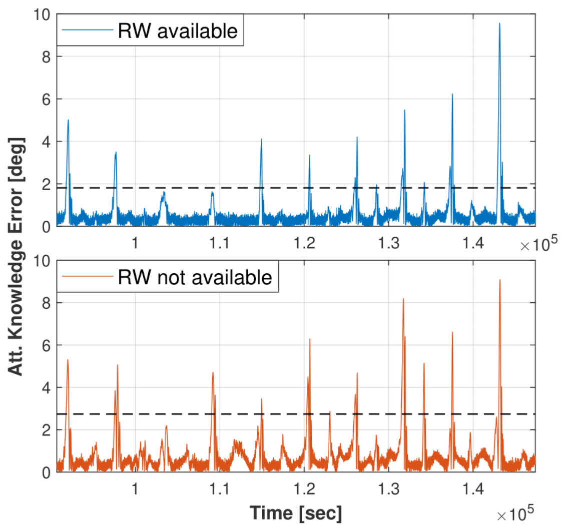

Figure 9: Absolute attitude knowledge error in *normal pointing*. The dashed line represents the TPM.

MBSE also enables the reuse of existing models, something that is made clear by the reuse of the model developed in [9], which served as a basis for the model developed in this work.

The framework developed for the integration of MBSE with simulation was successfully applied to the generation of a Functional Engineering Simulator. In fact, the model repository remained the single source of truth of all the specification used in the simulation model, except for the components internal modelling.

Model transformation approaches are currently capable of mapping the structural decomposition and connections between elements, specified in SysML, into simulation platform-specific semantics. The transfer of parameter values from SysML to Simulink can also be implemented robustly as long as the parameters used in the definition of the mathematical models are all modelled in SysML.

The least mature aspect of the model transformation approach is the capability of transforming behaviour specification defined in SysML into Simulink semantics. The behaviour modelled with activities can in general be decomposed down to the lowest level of the architecture. On the contrary, state-based and service-based behaviours, modelled respectively with state machines and *interactions*, are more difficult to decompose. Because of this, the capability of the model transformation framework to transform SysML state machines into Stateflow blocks in Simulink is essential to ensure that it is mature enough for use in an industry setting.

In the point of view of an user of this MBSE methodology, the use of the methods, tools, and languages mentioned in this work has a steep learning curve. Additionally, developing a system model takes a lot of time, and requires the systems engineer to be very thorough. However, these issues are compensated by: (i) the clear guidance to the process of system design provided by the MBSE methodology; (ii) the help provided by the MBSE tool to avoid making mistakes in the specification of the system; and (iii) the time saved in later stages of the system definition, due to the improved quality of the specification produced in earlier stages.

5. Conclusions

Model-based systems engineering has the potential to further improve the effectiveness of SE practices, by introducing modelling and information centralization into these. However, a few challenges must be first overcome, before MBSE may fulfil its promise. Two of these are the lack of practical guidance on the application of MBSE and the limitations on the integration of MBSE with simulation.

In this thesis, an MBSE methodology based on the OOSEM is proposed for the design of a subsystem, which includes a framework for integrating MBSE with simulation, through a model transformation approach. The methodology is demonstrated by being successfully applied to the design of an ADCS of a small satellite, where simulation is used to evaluate subsystem performance and compare alternative designs.

We conclude that an MBSE approach can be applied at the subsystem-level, ensuring a proper integration of systems engineering with specialty domain activities. A "subsystem engineer" is required, to integrate the two domains, but both the systems engineers and domain specialist engineers may still work with the types of models and methods to which they are accustomed. Model transformation approaches ensure information centralization and allow domain specialist engineers to perform mathematical modelling and simulation inside the scope of simulation environments. With this approach, the risk of a simulator infrastructure having to be reconfigured to be applied in the design of a system with unusual characteristics is avoided. As these infrastructures increase in complexity, reconfiguring them becomes harder and more time-consuming.

Acknowledgements

First, I would like to thank Professor Paulo Gil and João Paulo Monteiro for their ceaseless support during the development of this thesis, for the guidance provided, and, most of all, for all they have taught me regarding all kinds of subjects.

Also, I am grateful to Dassault Systemes for providing me the license to use the Cameo Systems Modeler, and to Mathworks Inc., for providing a licence of the MATLAB/Simulink software.

References

[1] Y. Cao, Y. Liu, and C. J. Paredis. System-Level Model Integration of Design and Simulation for Mechatronic Systems Based on SysML. *Mechatronics*, 21(6):1063–1075, 2011.

[2] B. Chabibi, A. Anwar, and M. Nassar. Towards a Model Integration from SysML to MATLAB/Simulink. *Journal of Software*, 13(12):630–645, 2018.

[3] A. Chulliat, W. Brown, P. Alken, C. Beggan, M. Nair, G. Cox, A. Woods, S. Macmillan, B. Meyer, and M. Paniccia. The US/UK World Magnetic Model for 2020-2025. Technical report, National Centers for Environmental Information, NOAA, 2020.

[4] ECSS Secretariat. Space Engineering - System Modelling and Simulation. Technical Report ECSS-E-TM-10-21A, ESA-ESTEC Requirements and Standards Division, Noordwijk, The Netherlands, 2010.

[5] J. P. Elm and D. R. Goldenson. Quantifying the Effectiveness of Systems Engineering. In *2013 IEEE International Systems Conference (SysCon)* (15-18 April 2013, Orlando, FL, USA), pages 6–13, 2013.

[6] J. Estefan. Survey of Model-Based Systems Engineering Methodologies, Rev. B. Technical report, NASA Jet Propulsion Lab, 2008.

[7] J. Fisher. Model-Based Systems Engineering: A New Paradigm. *INSIGHT*, 1(3):3–4, 1998.

[8] S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, New York, NY, USA, 2008. ISBN: 978-0-12-374379-4.

[9] S. Friedenthal and C. Oster. *Architecting Spacecraft with SysML: A Model-Based Systems Engineering Approach*. CreateSpace Independent Publishing Platform, 2017. ISBN: 978-1-54-428806-2.

[10] S. Gao, W. Cao, L. Fan, and J. Liu. MBSE for Satellite Communication System Architecting. *IEEE Access*, 7:51–67, 11 2019.

[11] J. Guo, E. Gill, and S. Figari. Model Based Systems Engineering to Support the Development of Nano-Satellites. In paper presented at the 65th International Astronautical Congress, 29 September - 3 October, Toronto, Canada, 2014.

[12] C. Hiep and M. Ioki. A Model-Based Systems Engineering (MBSE) Approach to Development of Attitude Determination and Control Subsystem for First Micro-Satellite in Vietnam. Master's thesis, Graduate School of Design and Management, Keio University, Vietnam, 2016.

[13] T. Huldt and I. Stenius. State-of-Practice Survey of Model-Based Systems Engineering. *Systems Engineering*, 22(2):134–145, 2019.

[14] F. Jakob, S. Mazzini, and A. Jung. A SysML-Based Methodology in a Concurrent Satellite Design Process. Technical Report 2011-01-2713, Society of Automotive Engineers, 2011.

[15] D. Kaslow, G. Soremekun, H. Kim, and S. Spangelo. Integrated Model-Based Systems Engineering (MBSE) Applied to the Simulation of a CubeSat Mission. In *2014 IEEE Aerospace Conference*, pages 1–14, 2014.

[16] R. Mahony, T. Hamel, and J.-M. Pflimlin. Nonlinear Complementary Filters on the Special Orthogonal Group. *IEEE Transactions on Automatic Control*, 53(5):1203–1218, 2008.

[17] J. P. Monteiro, R. M. Rocha, A. Silva, R. Afonso, and N. Ramos. Integration and Verification Approach of ISTSat-1 CubeSat. *Aerospace*, 6(12), 2019.

[18] C. Nigischer, S. Bougain, R. Riegler, H. P. Stanek, and M. Grafinger. Multi-Domain Simulation Utilizing SysML: State of the Art and Future Perspectives. *Procedia CIRP*, 100:319–324, 2021.

[19] Object Management Group (OMG). SysML Extension for Physical Interaction and Signal Flow Simulation, version 1.1, 2018. [Online; accessed 12-Abr-2021].

[20] Object Management Group (OMG). Omg systems modeling language specification, version 1.6, 2019. [Online; accessed 12-Abr-2021].

[21] A. G. Romero and M. Ferreira. Modeling an attitude and orbit control system using SysML. In paper presented at the 2nd Workshop in Space Technology and Engineering, 3-4 May, São Paulo, Brazil, 2011.

[22] M. L. Uhran. Mission accomplished! The role of systems engineering amp; Integration in the International Space Station program. In *2015 IEEE 26th Symposium on Fusion Engineering (SOFE)*, 2015.

[23] M. Waseem and M. U. Sadiq. Application of Model-Based Systems Engineering in Small Satellite Conceptual Design – A SysML Approach. *IEEE Aerospace and Electronic Systems Magazine*, 33(4):24–34, 2018.