

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

**Rodrigo Jorge Silva Ramos**

Thesis to obtain the Master of Science Degree in

**Aerospace Engineering**

Supervisors: Prof. Paulo Jorge Soares Gil  
Eng. João Paulo Lopes Monteiro

## **Examination Committee**

Chairperson: Prof. Afzal Suleman  
Supervisor: Prof. Paulo Jorge Soares Gil  
Member of the Committee: Dr. Loris Franchi

**December 2021**



## Acknowledgments

First and foremost, 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 for everything they have taught me regarding all kinds of subjects.

I would also like to recognize the importance of the ISTSat-1 project for the development of this work. Even though this thesis did not directly concern this mission, I was able to gain invaluable knowledge by participating in this project, which became very useful in the development of this work.

Furthermore, 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.

To my fellow colleagues from Instituto Superior Técnico, I would like them to know it was an honor to share this journey with all of them.

Finally, I wish to express my gratitude to my family and friends, for their love and support, in good and bad times alike. Without them this would not have been possible.



## Resumo

A engenharia de sistemas (SE) foi adotada na indústria aeroespacial para ajudar engenheiros a gerir complexidade. Contudo, à medida que os sistemas se estão a tornar mais distribuídos e interconectados, os mecanismos tradicionais que garantem a manutenção de consistência em SE — documentos escritos — estão-se a tornar gradualmente inadequados. Engenharia de sistemas baseada em modelos (MBSE) pode mitigar este problema, introduzindo modelação e centralização de informação nas práticas tradicionais de SE. Investigação tem mostrado que MBSE pode ser aplicado com sucesso em processos de design conceptual de sistemas. Contudo, quando aplicado a design de subsistemas, desafios diferentes são enfrentados relativamente à integração de SE com áreas de conhecimento especializadas. A integração de MBSE com simulação constitui um exemplo de um desafio à adoção destas práticas na indústria. Neste trabalho, uma metodologia baseada em OOSEM é proposta para o design de um subsistema de um veículo espacial. Esta inclui uma ferramenta de transformação de modelos capaz de gerar modelos de Simulink, a partir de especificação em SysML. A metodologia é demonstrada no design de um sistema de determinação e controlo de atitude de um pequeno satélite, com simulação sendo utilizada para avaliar performance e comparar designs alternativos. Com o trabalho desenvolvido, verificamos alguns benefícios de MBSE e concluímos que este tipo de abordagem pode ser aplicada ao nível de subsistema, integrando adequadamente engenharia de sistemas com áreas especializadas. Demonstramos ainda que abordagens de transformação de modelos são capazes de gerar simuladores completamente funcionais a partir de especificação em SysML.

**Palavras-chave:** MBSE, SysML, Transformação de modelos, OOSEM, MATLAB/Simulink, ADCS



## Abstract

Systems engineering (SE) has been adopted in the aerospace industry to help engineers manage complexity. However, as systems become more distributed and interconnected, the traditional mechanisms through which SE ensured consistency — written documents — are becoming increasingly inadequate. Model-based systems engineering (MBSE) can help mitigate this issue, by introducing modelling and information centralization into traditional SE practices. Some research has been done showing that MBSE can successfully be applied to system-level conceptual design processes. However, when applied to subsystem design, MBSE faces the challenge of integrating systems engineering with specialized knowledge domains. Specifically, the integration of MBSE with simulation presents itself as a major challenge to the adoption of MBSE practices in industry settings. In this work, a methodology based on the OOSEM is proposed for the design of a spacecraft subsystem, including a framework for integrating MBSE with simulation through a model transformation approach. System specification in SysML is transformed into a simulation model in Simulink, capable of evaluating system performance. 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. Some benefits of MBSE are verified, and we conclude that an MBSE approach can be applied at the subsystem-level, with a proper integration of systems engineering and specialized domains. 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





# Contents

|                                                                  |           |
|------------------------------------------------------------------|-----------|
| Acknowledgments . . . . .                                        | iii       |
| Resumo . . . . .                                                 | v         |
| Abstract . . . . .                                               | vii       |
| List of Tables . . . . .                                         | xiii      |
| List of Figures . . . . .                                        | xv        |
| Nomenclature . . . . .                                           | xvii      |
| Glossary . . . . .                                               | xxiii     |
| <b>1 Introduction</b>                                            | <b>1</b>  |
| 1.1 Objectives and Motivation . . . . .                          | 1         |
| 1.2 Space Systems Design and Systems Engineering . . . . .       | 1         |
| 1.2.1 Space Systems . . . . .                                    | 1         |
| 1.2.2 Systems Engineering . . . . .                              | 2         |
| 1.2.3 Motivation for Model-Based Systems Engineering . . . . .   | 3         |
| 1.2.4 Deployment of Model Based Systems Engineering . . . . .    | 5         |
| 1.2.5 Integration of MBSE with Simulation Tools . . . . .        | 6         |
| 1.3 Literature Review . . . . .                                  | 7         |
| 1.3.1 MBSE Deployment . . . . .                                  | 8         |
| 1.3.2 Integration of MBSE with Simulation Tools . . . . .        | 9         |
| 1.4 Research Contributions and Thesis Outline . . . . .          | 10        |
| <b>2 Theoretical Background</b>                                  | <b>11</b> |
| 2.1 MBSE Methodology Review . . . . .                            | 11        |
| 2.2 SysML . . . . .                                              | 13        |
| 2.2.1 Diagram Notation and Taxonomy . . . . .                    | 13        |
| 2.2.2 Basic elements of the language . . . . .                   | 14        |
| 2.3 Object-Oriented Systems Engineering Method (OOSEM) . . . . . | 17        |
| 2.4 Tool Selection and Integration . . . . .                     | 19        |
| 2.5 Design Process Definition . . . . .                          | 21        |
| 2.6 ADCS Design and technology . . . . .                         | 23        |
| 2.6.1 Sensors . . . . .                                          | 24        |

|          |                                                         |           |
|----------|---------------------------------------------------------|-----------|
| 2.6.2    | Attitude Determination . . . . .                        | 25        |
| 2.6.3    | Actuators . . . . .                                     | 26        |
| 2.6.4    | Attitude Guidance and Control . . . . .                 | 27        |
| <b>3</b> | <b>Approach</b>                                         | <b>28</b> |
| 3.1      | Methodology Adaptation . . . . .                        | 28        |
| 3.1.1    | Process Adaptation . . . . .                            | 29        |
| 3.1.2    | Activity Definition . . . . .                           | 30        |
| 3.1.3    | Integrating MBSE with Simulation . . . . .              | 32        |
| 3.2      | Model Setup . . . . .                                   | 41        |
| 3.3      | Methodology Validation . . . . .                        | 42        |
| <b>4</b> | <b>Methodology Demonstration</b>                        | <b>43</b> |
| 4.1      | Design Process Inputs . . . . .                         | 43        |
| 4.1.1    | Mission-level inputs . . . . .                          | 43        |
| 4.1.2    | System-level inputs . . . . .                           | 44        |
| 4.1.3    | Subsystem-level inputs . . . . .                        | 46        |
| 4.2      | Subsystem Design . . . . .                              | 50        |
| 4.2.1    | Subsystem State Machine Definition . . . . .            | 51        |
| 4.2.2    | Functional Architecture Definition . . . . .            | 52        |
| 4.2.3    | Logical Architecture Design . . . . .                   | 56        |
| 4.2.4    | Synthesis of Candidate Physical Architectures . . . . . | 63        |
| 4.2.5    | Management of Requirement Traceability . . . . .        | 67        |
| 4.2.6    | Evaluation of Alternatives . . . . .                    | 68        |
| 4.3      | Design Process Outputs . . . . .                        | 75        |
| 4.4      | Discussion of Results . . . . .                         | 76        |
| <b>5</b> | <b>Conclusions</b>                                      | <b>79</b> |
|          | <b>Bibliography</b>                                     | <b>79</b> |
| <b>A</b> | <b>Spacecraft Attitude</b>                              | <b>91</b> |
| A.1      | Reference Frames . . . . .                              | 91        |
| A.1.1    | Inertial Reference Frame . . . . .                      | 91        |
| A.1.2    | Earth-Centred/Earth-Fixed Frame . . . . .               | 91        |
| A.1.3    | Topocentric Horizon Coordinate System (NED) . . . . .   | 93        |
| A.1.4    | Spacecraft Body Frame . . . . .                         | 93        |
| A.2      | Attitude Representation . . . . .                       | 94        |
| A.2.1    | Attitude Matrix . . . . .                               | 94        |
| A.2.2    | Quaternions . . . . .                                   | 94        |
| A.3      | Spacecraft Attitude Dynamics . . . . .                  | 96        |

|          |                                                     |            |
|----------|-----------------------------------------------------|------------|
| A.3.1    | External disturbance torques . . . . .              | 96         |
| <b>B</b> | <b>ADCS Requirements Specification</b>              | <b>98</b>  |
| B.1      | Specification of Performance Requirements . . . . . | 98         |
| B.2      | Complete List of ADCS Requirements . . . . .        | 100        |
| <b>C</b> | <b>Spacecraft Environment and Orbital Dynamics</b>  | <b>102</b> |
| C.1      | Expansion in Spherical Harmonics . . . . .          | 102        |
| C.2      | Earth Magnetic Field . . . . .                      | 103        |
| C.3      | Earth Gravity Field . . . . .                       | 103        |
| C.4      | Earth Albedo . . . . .                              | 104        |
| <b>D</b> | <b>Mathematical Models of Components</b>            | <b>106</b> |
| D.1      | Hardware . . . . .                                  | 106        |
| D.1.1    | Sensors . . . . .                                   | 106        |
| D.1.2    | Actuators . . . . .                                 | 107        |
| D.2      | Software . . . . .                                  | 107        |
| D.2.1    | Estimation Algorithms . . . . .                     | 107        |
| D.2.2    | Control Algorithms . . . . .                        | 110        |



# List of Tables

|     |                                                                                                                                                                                                                                           |     |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 2.1 | Comparison of the reviewed methodologies' key aspects . . . . .                                                                                                                                                                           | 12  |
| 2.2 | Mapping between SysML and Simulink/Stateflow constructs specified by the OMG SysPhS. . . . .                                                                                                                                              | 20  |
| 3.1 | Mapping between SysML state machine and Stateflow constructs. . . . .                                                                                                                                                                     | 37  |
| 4.1 | Definition of mission attributes. The entry orbit is the same as for the ISTSat-1: Sun-Synchronous Orbit (SSO) with LTAN at 22:30. . . . .                                                                                                | 44  |
| 4.2 | Definition of relevant system attributes. . . . .                                                                                                                                                                                         | 46  |
| 4.3 | Critical properties of the Earth sensor, gyroscope, and magnetometer. . . . .                                                                                                                                                             | 64  |
| 4.4 | Critical properties of the reaction wheel, magnetic torquer, and photodiode. . . . .                                                                                                                                                      | 64  |
| 4.5 | Critical properties of the estimators of the magnetic field and spacecraft and Sun positions. . . . .                                                                                                                                     | 65  |
| 4.6 | Critical properties of the attitude filter and attitude controller. . . . .                                                                                                                                                               | 66  |
| 4.7 | Configurations used in GMAT for the determination of the spacecraft orbit. . . . .                                                                                                                                                        | 70  |
| 4.8 | Schedule of events representing the simulation scenario defined. . . . .                                                                                                                                                                  | 72  |
| 4.9 | Summary of the simulation results (the architecture with reaction wheels is named "A1", and the other "A2"). The parameter values obtained with simulation (left) are compared to maximum values imposed by requirements (right). . . . . | 73  |
| B.1 | ADCS Functional Requirements. . . . .                                                                                                                                                                                                     | 100 |
| B.2 | ADCS Operational Requirements. . . . .                                                                                                                                                                                                    | 101 |
| B.3 | ADCS Performance Requirements. . . . .                                                                                                                                                                                                    | 101 |
| B.4 | ADCS Design Constraints. . . . .                                                                                                                                                                                                          | 101 |



# List of Figures

|      |                                                                                                                                                                                                                                                                  |    |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1  | Example of elements of a system model contained inside a model repository. . . . .                                                                                                                                                                               | 4  |
| 2.1  | (a) Representation of a generic block and its features. (b) Representation of a <i>generalization</i> relationship, on the top, and a <i>composite association</i> , on the bottom. (c) Parametric representation of the internal structure of a block . . . . . | 15 |
| 2.2  | Representation of the most common relationships involving requirements. . . . .                                                                                                                                                                                  | 16 |
| 2.3  | “Vee” model of the SOI life cycle. . . . .                                                                                                                                                                                                                       | 17 |
| 2.4  | OOSEM process (each node represents an activity). . . . .                                                                                                                                                                                                        | 18 |
| 2.5  | Functional Engineering Simulator (FES) structure (Figure 6 in [60]). . . . .                                                                                                                                                                                     | 19 |
| 2.6  | Definition of target scope using the “Vee” model. . . . .                                                                                                                                                                                                        | 21 |
| 2.7  | Architecture levels defined in this work. . . . .                                                                                                                                                                                                                | 22 |
| 2.8  | Black-box representation of the subsystem design process. . . . .                                                                                                                                                                                                | 23 |
| 2.9  | Structure of an ADCS based on active control techniques. . . . .                                                                                                                                                                                                 | 24 |
| 3.1  | Subsystem design methodology process (each node represents an activity or task). . . .                                                                                                                                                                           | 29 |
| 3.2  | <i>Manage Requirements Traceability</i> process (each node represents a task). . . . .                                                                                                                                                                           | 30 |
| 3.3  | <i>Define Functional Architecture</i> process (each node represents a task). . . . .                                                                                                                                                                             | 31 |
| 3.4  | <i>Define Logical Architecture</i> process (each node represents a task). . . . .                                                                                                                                                                                | 31 |
| 3.5  | <i>Synthesize Candidate Physical Architectures</i> process (each node represents a task). . .                                                                                                                                                                    | 32 |
| 3.6  | <i>Optimize and Evaluate Alternatives</i> process (each node represents a task). . . . .                                                                                                                                                                         | 32 |
| 3.7  | Model transformation workflow. The large grey arrows represent steps in the transformation.                                                                                                                                                                      | 34 |
| 3.8  | Example of the mapping between SysML and Simulink representations of a component.<br>The environment is also divided into components, which are identified with the <i>stereotype</i> <i>«analysis»</i> . . . . .                                                | 36 |
| 3.9  | Segment of the table outlining the <i>part properties</i> relevant to the simulation model. . . .                                                                                                                                                                | 36 |
| 3.10 | Segment of the table outlining the values defined with <i>instance specifications</i> . . . . .                                                                                                                                                                  | 37 |
| 3.11 | List of state machines relevant to the simulation model (left) and correspondence between<br>each state machine and the name of the variable that stores its active state (right). . . .                                                                         | 38 |
| 3.12 | Segment of the table outlining the states contained in the relevant state machines. . . .                                                                                                                                                                        | 38 |
| 3.13 | Example of a Stateflow block generated from SysML specification. . . . .                                                                                                                                                                                         | 39 |
| 3.14 | Segment of the table outlining the transitions contained in the relevant state machines. . .                                                                                                                                                                     | 39 |

|                                                                                                                                                                                                            |    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.15 Model Organization. . . . .                                                                                                                                                                           | 42 |
| 4.1 Mission architecture. . . . .                                                                                                                                                                          | 43 |
| 4.2 System decomposition. . . . .                                                                                                                                                                          | 44 |
| 4.3 Mission- and system-level context of the ADCS. . . . .                                                                                                                                                 | 45 |
| 4.4 Spacecraft state machine behaviour specification. . . . .                                                                                                                                              | 45 |
| 4.5 <i>Specification tree</i> (only the part relevant to the ADCS). . . . .                                                                                                                                | 47 |
| 4.6 Example of traceability between mission-level, system-level, and ADCS requirements. . . . .                                                                                                            | 48 |
| 4.7 ADCS black box specification. . . . .                                                                                                                                                                  | 50 |
| 4.8 Representation of the autonomous transitions originated in the <i>detumbling</i> mode. . . . .                                                                                                         | 51 |
| 4.9 Representation of the autonomous transitions originated in the pointing modes. . . . .                                                                                                                 | 52 |
| 4.10 Representation of the forced transitions and state machine initialization. . . . .                                                                                                                    | 52 |
| 4.11 Functional decomposition of the subsystem. . . . .                                                                                                                                                    | 53 |
| 4.12 Interaction between functional components realizing the <i>detumbling operation activity</i> . . . . .                                                                                                | 54 |
| 4.13 Interaction between functional components realizing the <i>normal pointing operation activity</i> . . . . .                                                                                           | 55 |
| 4.14 Interaction between functional components realizing the <i>safe pointing operation activity</i> . . . . .                                                                                             | 55 |
| 4.15 Internal structure of the ADCS functional architecture. . . . .                                                                                                                                       | 56 |
| 4.16 Logical architecture decomposition and allocation of functional to logical components. . . . .                                                                                                        | 57 |
| 4.17 Interaction between logical components realizing the <i>detumbling operation activity</i> . . . . .                                                                                                   | 59 |
| 4.18 Interaction between logical components realizing the <i>normal pointing operation activity</i> . . . . .                                                                                              | 59 |
| 4.19 Interaction between logical components realizing the <i>safe pointing operation activity</i> . . . . .                                                                                                | 60 |
| 4.20 Internal structure of the ADCS logical architecture. . . . .                                                                                                                                          | 61 |
| 4.21 Redefinition of the external context of the ADCS. . . . .                                                                                                                                             | 62 |
| 4.22 Specification of which elements in the model satisfy and verify each requirement. . . . .                                                                                                             | 68 |
| 4.23 Internal structure of the <i>Environment</i> block. . . . .                                                                                                                                           | 69 |
| 4.24 Internal structure of the <i>analysis context</i> and the <i>Monitoring and Control</i> block. . . . .                                                                                                | 71 |
| 4.25 Specification of the simulation scenario with a schedule of <i>signal events</i> . . . . .                                                                                                            | 73 |
| 4.26 Absolute attitude knowledge error during the 10 orbits of performance evaluation for the <i>normal pointing</i> mode. The dashed line is located at the final value of the corresponding TPM. . . . . | 74 |
| 4.27 Absolute pointing error during the 10 orbits of performance evaluation for the <i>normal pointing</i> mode. The dashed line is located at the final value of the corresponding TPM. . . . .           | 75 |
| 4.28 Absolute pointing error during the 10 orbits of performance evaluation for the <i>safe pointing</i> mode. The dashed line is located at the final value of the corresponding TPM. . . . .             | 75 |
| A.1 Spacecraft body frame and nomenclature used to describe faces. . . . .                                                                                                                                 | 93 |



# Nomenclature

## Roman symbols

|                   |                                                                                                       |
|-------------------|-------------------------------------------------------------------------------------------------------|
| $\check{P}_{n,m}$ | Schmidt semi-normalized associated Legendre functions of degree $n$ and order $m$ .                   |
| $\tilde{q}$       | Attitude quaternion estimation error.                                                                 |
| $\mathbf{0}_3$    | Null vector representation.                                                                           |
| $\delta q$        | Control error quaternion.                                                                             |
| $a$               | Spacecraft acceleration vector.                                                                       |
| $a_e$             | Euler axis vector.                                                                                    |
| $B$               | Local Earth magnetic field vector.                                                                    |
| $b$               | Local Earth magnetic field direction unit vector (always represented in the body frame).              |
| $b_g$             | Gyroscope measurement bias.                                                                           |
| $b_m$             | Magnetometer measurement bias.                                                                        |
| $e$               | Basis vectors of an arbitrary reference frame.                                                        |
| $F$               | Force vector.                                                                                         |
| $f$               | Representation of the derivative of $X$ relative to time.                                             |
| $F_{\text{aero}}$ | Drag force applied in one spacecraft face (always represented in the body frame).                     |
| $g$               | Acceleration of Earth's gravity.                                                                      |
| $H$               | Spacecraft angular momentum (always represented in the body frame).                                   |
| $h$               | Angular momentum stored by rotating objects of the spacecraft (always represented in the body frame). |
| $I_q$             | Identity Quaternion.                                                                                  |
| $m_{\text{mp}}$   | Commanded magnetic dipole for magnetic torquer-based pointing.                                        |
| $m_d$             | Commanded magnetic dipole for detumbling.                                                             |

|                            |                                                                                                 |
|----------------------------|-------------------------------------------------------------------------------------------------|
| $\mathbf{m}_u$             | Spacecraft undesired magnetic dipole.                                                           |
| $\mathbf{n}$               | Local nadir direction.                                                                          |
| $\mathbf{n}_{\text{mtq}}$  | Direction of the magnetic dipole produced by one magnetic torquer.                              |
| $\mathbf{n}_{\text{pld}}$  | Vector in the body frame that must be aligned with nadir (payload direction).                   |
| $\mathbf{n}_c$             | Outward normal of a grid cell (Earth albedo model).                                             |
| $\mathbf{n}_s$             | Outward normal unit vector of a face/plate (always represented in the body frame).              |
| $\mathbf{n}_w$             | Reaction wheel axis of rotation.                                                                |
| $\mathbf{p}$               | Representation of an arbitrary quaternion.                                                      |
| $\mathbf{q}$               | Attitude quaternion (without subscripts, represents transformation from $I$ to $B$ ).           |
| $\mathbf{r}$               | Position vector of the spacecraft.                                                              |
| $\mathbf{r}_{\text{Sun}}$  | Position of the Sun in the ECI frame.                                                           |
| $\mathbf{r}_p$             | Position of the centre of pressure of a spacecraft face (always represented in the body frame). |
| $\mathbf{s}$               | Estimation/measurement of a given known vector quantity.                                        |
| $\mathbf{T}$               | Sum of external torques applied in the spacecraft (always represented in the body frame).       |
| $\mathbf{T}_{\text{mtq}}$  | Actuation torque produced by one magnetic torquer.                                              |
| $\mathbf{T}_w$             | Actuation torque produced by one reaction wheel.                                                |
| $\mathbf{T}_{\text{aero}}$ | Aerodynamic disturbance torque (always represented in the body frame).                          |
| $\mathbf{T}_{\text{gg}}$   | Gravity-gradient torque (always represented in the body frame).                                 |
| $\mathbf{T}_m$             | Magnetic disturbance torque (always represented in the body frame).                             |
| $\mathbf{u}$               | Representation of an arbitrary vector.                                                          |
| $\mathbf{u}_{\text{Sun}}$  | Sun vector in the body frame                                                                    |
| $\mathbf{v}$               | Velocity vector of the spacecraft.                                                              |
| $\mathbf{v}_{\text{rel}}$  | Velocity of the spacecraft relative to the air.                                                 |
| $\mathbf{X}$               | State vector for the equations of motion (see Eq. D.8).                                         |
| $\mathbf{A}$               | Attitude matrix (without subscripts, represents transformation from $I$ to $B$ ).               |
| $a$                        | Semi-major axis.                                                                                |
| $\mathbf{C}$               | Representation of an arbitrary rotation matrix.                                                 |
| $C_{n,m}$                  | Normalized geopotential coefficients.                                                           |

|              |                                                                                   |
|--------------|-----------------------------------------------------------------------------------|
| $C_D$        | Coefficient of drag.                                                              |
| $C_q$        | Rotation matrix derived from an unit quaternion.                                  |
| $C_r$        | Coefficient of reflectivity.                                                      |
| $D$          | Commanded dutycycle of one magnetic torquer.                                      |
| $D_{\max}$   | Maximum dutycycle that can be produced by one magnetic torquer.                   |
| $E$          | Irradiance.                                                                       |
| $e$          | Orbit eccentricity.                                                               |
| $E_{AM0}$    | Solar irradiance at Earth.                                                        |
| $e_p$        | Performance requirement error function.                                           |
| $g_{n,m}$    | Gauss coefficients.                                                               |
| $h_{n,m}$    | Gauss coefficients.                                                               |
| $h_w$        | Angular momentum stored by one reaction wheel.                                    |
| $i$          | Orbit inclination.                                                                |
| $I(e_p)$     | Performance requirement error Index.                                              |
| $I_{\max}$   | Upper bound for the error index.                                                  |
| $I_{TPM}$    | TPM value.                                                                        |
| $i_k$        | Current output of photodiode $k$ .                                                |
| $I_s$        | Moment of inertia of the spacecraft measured in the body frame.                   |
| $I_w$        | Reaction wheel moment of inertia.                                                 |
| $k_{mm}$     | Reaction wheel off-loading positive scalar gain.                                  |
| $k_{pd}$     | Photodiode conversion factor.                                                     |
| $k_d$        | Detumbling control positive scalar gain.                                          |
| $k_g$        | Bias correction positive scalar gain.                                             |
| $k_i$        | Positive scalar gain for the $i^{th}$ sensor correction.                          |
| $k_p$        | Correction factor positive scalar gain.                                           |
| $k_r$        | Angular velocity rotation positive scalar gain (magnetic torquer-based pointing). |
| $k_\epsilon$ | Proportional positive scalar gain.                                                |
| $k_\omega$   | Derivative positive scalar gain.                                                  |

|                   |                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------|
| $M$               | Mean anomaly.                                                                                    |
| $m_{\max}$        | Maximum magnetic dipole that can be produced by one magnetic torquer.                            |
| $P_C$             | Required probability that the index stays between bounds (concerning a performance requirement). |
| $P_c$             | Incident radiant flux for one cell grid.                                                         |
| $P_{n,m}$         | Associated Legendre polynomial of degree $n$ and order $m$ .                                     |
| $R$               | Inverse/transpose of the attitude matrix.                                                        |
| $r$               | Magnitude of the position vector of the spacecraft.                                              |
| $R_{\oplus}$      | Earth radius.                                                                                    |
| $S$               | Area of a face/plate.                                                                            |
| $S_{n,m}$         | Normalized geopotential coefficients.                                                            |
| $S_c$             | Area of a grid cell (Earth albedo model).                                                        |
| $S_{\text{drag}}$ | Drag reference area.                                                                             |
| $S_{\text{SRP}}$  | SRP reference area.                                                                              |
| $t$               | Representation of time.                                                                          |
| $T_0$             | Instant of spacecraft separation from launcher.                                                  |
| $U$               | Gravitational potential.                                                                         |
| $V$               | Magnetic Field Scalar Potential.                                                                 |
| $x, y, z$         | Position cartesian components.                                                                   |

### Greek symbols

|                      |                                                                             |
|----------------------|-----------------------------------------------------------------------------|
| $\beta_g$            | Filter bias correction.                                                     |
| $\gamma$             | Filter static correction factor.                                            |
| $\epsilon$           | Desired rotation vector (magnetic torquer-based pointing).                  |
| $\epsilon_{\omega}$  | Desired angular velocity rotation vector (magnetic torquer-based pointing). |
| $\mu_{\text{arw}}$   | White noise process known as Angle Random Walk.                             |
| $\mu_{\text{m}}$     | Magnetometer noise component.                                               |
| $\mu_{\text{rrw}}$   | White noise process known as Rate Random Walk.                              |
| $\tau_{\text{mp,d}}$ | Desired control torque for magnetic torquer-based pointing.                 |

|                    |                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------|
| $\tau_{mp}$        | Actual control torque for magnetic torquer-based pointing.                                              |
| $\tau_d$           | Actual control torque in detumbling.                                                                    |
| $\tau_{mm,d}$      | Desired torque for reaction wheel off-loading.                                                          |
| $\tau_{mm}$        | Actual torque for reaction wheel off-loading produced by the magnetic torquers.                         |
| $\tau_{rw}$        | Desired control torque for reaction wheel-based stabilization.                                          |
| $\omega$           | Angular velocity of the spacecraft (always represented in the body frame).                              |
| $\omega_c$         | Commanded angular velocity.                                                                             |
| $\omega_w$         | Sum of the angular velocities of the three reaction wheels.                                             |
| $\Delta T$         | Time passed since separation up to the current instant, expressed in seconds.                           |
| $\epsilon_{es}$    | Earth sensor angular error noise process.                                                               |
| $\theta_{aero}$    | Angle between the normal of a spacecraft's face and the velocity of the spacecraft relative to the air. |
| $\theta_{GMST}$    | Greenwich Mean Sidereal Time angle.                                                                     |
| $\lambda$          | Longitude.                                                                                              |
| $\mu_{\oplus}$     | Standard gravitational parameter of Earth.                                                              |
| $\nu$              | Euler angle.                                                                                            |
| $\Xi(\mathbf{q})$  | Matrix defined in Eq. A.15.                                                                             |
| $\rho$             | Air Density.                                                                                            |
| $\rho_c$           | Reflectivity of a grid cell (Earth albedo model).                                                       |
| $\phi$             | Geodetic latitude.                                                                                      |
| $\phi'$            | Geocentric latitude.                                                                                    |
| $\Psi(\mathbf{q})$ | Matrix defined in Eq. A.14.                                                                             |
| $\Omega$           | Right ascension of the ascending node (RAAN).                                                           |
| $\omega_p$         | Argument of perigee.                                                                                    |
| $\omega_w$         | Angular speed of one reaction wheel.                                                                    |
| $\omega_{\oplus}$  | Earth angular rotation speed.                                                                           |

### Subscripts

|   |                |
|---|----------------|
| 0 | Initial value. |
|---|----------------|

|      |                                                                                   |
|------|-----------------------------------------------------------------------------------|
| aero | Relative to the calculation of aerodynamic torque.                                |
| corr | Estimation that results from the correction of a previous estimation/measurement. |
| $B$  | Represented in the body reference frame.                                          |
| $E$  | Represented in the ECEF reference frame.                                          |
| $E'$ | Represented in the ECEF reference frame realized at $T_0$ .                       |
| $H$  | Represented in the NED reference frame.                                           |
| $I$  | Represented in the ECI reference frame.                                           |
| $xy$ | Projection of a vector onto the $Oxy$ plane.                                      |

### **Superscripts**

|      |                           |
|------|---------------------------|
| true | True value of a quantity. |
| *    | Conjugate.                |
| T    | Transpose.                |

# Acronyms

|                |                                                     |
|----------------|-----------------------------------------------------|
| <b>ADCS</b>    | Attitude Determination and Control System.          |
| <b>AIT</b>     | Assembly, Integration, and Testing.                 |
| <b>AKE</b>     | Absolute Knowledge Error.                           |
| <b>AOCS</b>    | Attitude and Orbit Control System.                  |
| <b>APE</b>     | Absolute Performance Error.                         |
| <b>API</b>     | Application Programming Interface.                  |
| <b>ARCADIA</b> | Architecture Analysis & Design Integrated Approach. |
| <b>ARW</b>     | Angle Random Walk.                                  |
| <b>ASR</b>     | Atmospheric Sample Return.                          |
|                |                                                     |
| <b>CDF</b>     | Concurrent Design Facility.                         |
| <b>CMG</b>     | Control Moment Gyro.                                |
| <b>ConOps</b>  | Concept of Operations.                              |
| <b>CPS</b>     | Cyber-Physical Systems.                             |
| <b>CSM</b>     | Cameo Systems Modeler.                              |
| <b>CSS</b>     | Coarse Sun Sensor.                                  |
| <b>CVG</b>     | Coriolis Vibrating Gyros.                           |
|                |                                                     |
| <b>DBSE</b>    | Document-Based Systems Engineering.                 |
| <b>DCM</b>     | Direction Cosine Matrix.                            |
| <b>DSS</b>     | Digital Sun Sensor.                                 |
|                |                                                     |
| <b>ECEF</b>    | Earth-Centred/Earth-Fixed.                          |
| <b>ECF</b>     | Explicit Complementary Filter.                      |
| <b>ECI</b>     | Earth-Centred Inertial.                             |
| <b>ECSS</b>    | European Cooperation for Space Standardization.     |
| <b>EGSE</b>    | Electrical Ground Support Equipment.                |
| <b>ESA</b>     | European Space Agency.                              |
|                |                                                     |
| <b>FES</b>     | Functional Engineering Simulator.                   |

|              |                                                                  |
|--------------|------------------------------------------------------------------|
| <b>FOV</b>   | Field of View.                                                   |
| <b>GCRF</b>  | Geocentric Celestial Reference Frame.                            |
| <b>GMST</b>  | Greenwich Mean Sidereal Time.                                    |
| <b>IBM</b>   | International Business Machines Corporation.                     |
| <b>ICRF</b>  | International Celestial Reference Frame.                         |
| <b>IEC</b>   | International Eletroctechanical Commission.                      |
| <b>IEEE</b>  | Institute of Electrical and Electronics Engineers.               |
| <b>ISO</b>   | International Standardization Organization.                      |
| <b>IT</b>    | Information Technology.                                          |
| <b>ITRF</b>  | International Terrestrial Reference Frame.                       |
| <b>LEO</b>   | Low Earth Orbit.                                                 |
| <b>LTAN</b>  | Local Time of the Ascending Node.                                |
| <b>MAF</b>   | Moving Average Filter.                                           |
| <b>MBSE</b>  | Model-Based Systems Engineering.                                 |
| <b>MBSSE</b> | Model-Based methodology to support the Space System Engineering. |
| <b>MEMS</b>  | Microelectronic Mechanical Systems.                              |
| <b>MOE</b>   | Measures of Effectiveness.                                       |
| <b>MOP</b>   | Measures of Performance.                                         |
| <b>NASA</b>  | National Aeronautics and Space Administration.                   |
| <b>NED</b>   | North-East-Down.                                                 |
| <b>OBC</b>   | On-board Computer.                                               |
| <b>OMG</b>   | Object Management Group.                                         |
| <b>OO</b>    | Object-oriented.                                                 |
| <b>OOSEM</b> | Object-Oriented Systems Engineering Method.                      |
| <b>OPM</b>   | Object-Process Methodology.                                      |
| <b>PD</b>    | Proportional-derivative.                                         |
| <b>PSD</b>   | Power Spectral Density.                                          |
| <b>PWM</b>   | Pulse-Width Modulation.                                          |
| <b>QVT</b>   | OMG Query-View-Transform.                                        |
| <b>RAAN</b>  | Right Ascension the Ascending Node.                              |



|                |                                                            |
|----------------|------------------------------------------------------------|
| <b>ROI</b>     | Return on Investment.                                      |
| <b>RRW</b>     | Rate Random Walk.                                          |
| <b>RVTM</b>    | Requirements Verification and Traceability Matrices.       |
| <b>SE</b>      | Systems Engineering.                                       |
| <b>SESP</b>    | Simulation and EGSE for Space Programmes (conference).     |
| <b>SEZ</b>     | South-East-Zenith.                                         |
| <b>SOI</b>     | System of Interest.                                        |
| <b>SOS</b>     | System-of-systems.                                         |
| <b>SRP</b>     | Solar Radiation Pressure.                                  |
| <b>SSO</b>     | Sun-Synchronous Orbit.                                     |
| <b>SysML</b>   | OMG Systems Modelling Language.                            |
| <b>SysPhS</b>  | OMG SysML Physical Interaction and Signal Flow Simulation. |
| <b>TPM</b>     | Technical Performance Measures.                            |
| <b>UTC</b>     | Universal Time Coordinated.                                |
| <b>V&amp;V</b> | Verification and Validation.                               |
| <b>VSD</b>     | Virtual Spacecraft Design.                                 |
| <b>WMM</b>     | World Magnetic Model.                                      |
| <b>XMI</b>     | XML Metadata Interchange.                                  |
| <b>XML</b>     | Extensible Markup Language.                                |

# Glossary

|                             |                                                                                                                                                                             |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>(MBSE) Methodology</b>   | Collection of related processes, methods, and tools used to support the systems engineering effort [62].                                                                    |
| <b>Body attitude</b>        | The three-dimensional orientation of the body with respect to a specified reference frame [22].                                                                             |
| <b>Co-simulation</b>        | Integration of multiple simulators to perform a composite simulation, involving weaving the time series behaviour and data exchanges accurately [36].                       |
| <b>Complexity</b>           | Measure of how difficult it is to understand how a system will behave [2].                                                                                                  |
| <b>CubeSat</b>              | Type of standardized nanosatellite composed of one or more units, with each unit being shaped as a 10 cm cube with a mass of up to 1.33 kg [83].                            |
| <b>Emergent Behaviour</b>   | “Behaviour of the system that cannot be understood exclusively in terms of the behaviour of the individual system elements” [3].                                            |
| <b>Iteration</b>            | Repeated application and interaction of processes at a given level in a system hierarchy [3].                                                                               |
| <b>Life Cycle</b>           | “The evolution with time of a SOI from conception to retirement” [3].                                                                                                       |
| <b>MBSE Modelling Tool</b>  | Class of tools that comply with the rules of the modelling language and are capable of creating, maintaining, and displaying the system model [15].                         |
| <b>Model Transformation</b> | Approach to semantic interoperability which establishes a correspondence between the constructs in one model and the constructs in another [3].                             |
| <b>Pseudostate</b>          | In the context of SysML, a pseudostate is similar to a state, but in which a state machine can never rest. “It merely exists to help determine the next active state” [14]. |

|                     |                                                                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Recursion</b>    | Repeated application and interaction of processes at successive levels in the system hierarchy [3].                                                                                                           |
| <b>Stakeholder</b>  | “A party having a right, share, or claim in a system or in its possession of characteristics that meet that party’s needs and expectations” [3].                                                              |
| <b>System Model</b> | An interconnected set of elements that represent requirements, design elements, test cases, and design rationale. It includes information relative to specification, design, analysis, and verification [14]. |
| <b>Validation</b>   | “Set of activities ensuring that a system is able to accomplish its intended use, goals, and objectives in the intended operational environment” [3].                                                         |
| <b>Verification</b> | “Set of activities that compares a system against the required characteristics” [3].                                                                                                                          |



# Chapter 1

## Introduction

### 1.1 Objectives and Motivation

The last decades have seen an increasing demand for high-performance multidisciplinary systems that deliver increased functionality with low development budgets and tight schedules. As a result, systems are becoming more complex, and Systems Engineering (SE) practices are being adopted to help engineers manage this complexity. SE is improving the understanding of stakeholder expectations and the ability of engineers to develop systems that function according to requirements.

Traditional SE approaches require expensive documentation management, which is an issue that Model-Based Systems Engineering (MBSE) promises to mitigate. By introducing modelling into SE practices and centralizing information into a system model, MBSE promises enhanced communication, improved quality, reduced development risk, and increased productivity.

In this thesis, an MBSE methodology is applied to the design of a spacecraft subsystem, namely an Attitude Determination and Control System (ADCS). The design must be supported by simulation, and this is accomplished by integrating an MBSE modelling tool with a simulation environment.

MBSE is still relatively unexplored in research, and it is still not clear how the interaction between the SE effort and the work developed in specialty domains can be performed. This work provides some insight into this interaction, specifically regarding the integration of MBSE with simulation, and into whether MBSE can support the design of a spacecraft subsystem and deliver the promised benefits.

### 1.2 Space Systems Design and Systems Engineering

#### 1.2.1 Space Systems

In this work, the term “space system” is used as a synonym of “space segment”, such as to refer to a spacecraft as opposed to ground systems. Space systems are designed to operate in extreme environments that are commonly difficult to recreate on Earth’s surface, making the complete verification of the system often impossible. Most times, failure must be avoided at all costs, resulting in high-reliability

constraints. These systems must also operate according to high-performance requirements usually due to tight mass, power, and volume budgets, which are often the main drivers of the mission cost. These characteristics and the low number of production units make these engineering systems unique [1].

In fact, space systems are in general complex engineering systems. A “complex system” should not be confused with a “complicated system”, i.e., one composed of a many parts and interfaces. Complexity is instead a measure of how difficult it is to understand how a system will behave [2]. The complexity of a system gives rise to the existence of emergent behaviours, i.e., “behaviours of the system that cannot be understood exclusively in terms of the behaviour of the individual system elements” [3, p. 6]. These behaviours are usually difficult to detect and understand and are a common source of failure to meet system requirements.

The high-performance and high-reliability requirements coupled with the need to deliver a variety of functions naturally results in high complexity. To illustrate the need for a variety of functions, we may consider an Earth orbiting satellite, which must at least: (i) communicate with the ground; (ii) generate, store, and distribute electric energy; (iii) control its attitude; (iv) manage its temperature; and (v) operate a payload (often the most complex subsystem). These are multidisciplinary functions that must be attributed to different subsystems. Also, the spacecraft and each subsystem usually have different modes of operation, to deliver the required functions while dealing with a changing environment. These two factors and the implementation of redundancy in the elements of the system favour the presence of emergent behaviours.

### **1.2.2 Systems Engineering**

The discipline known as Systems Engineering (SE) has been dominant in the aerospace and defence industry as an approach to deal with systems complexity, helping engineers to design systems meeting the requirements effectively and efficiently. In [4, p. 3], SE is defined as “the art and science of developing an operable system that meets requirements within imposed constraints”. SE aims to define and maintain full logical linking between the problem specification, the defined solution, verification and validation (V&V), and operations of a system of interest (SOI) [5]. It includes the application of both management and technical processes across the complete life cycle of the SOI and is usually implemented according to consolidated standards which define and specify a set of processes (e.g., ECSS [6], ISO/IEC 15288 [7]). The standards define what to be done at each stage, the inputs that are required, and the outputs to be produced. In this work, we focus on the technical processes of SE, which are applied to specify, design, and verify a SOI, and we discuss SE as applied to the development of space systems.

The adoption of SE practices requires an upfront investment of time and resources, so it is worth to first evaluate the real value of SE. In the absence of reliable industry benchmarks, the survey presented in [8] provides quantitative evidence about the effectiveness of SE. For the entire set of projects considered, the fraction of projects delivering higher performance increased from 15% to 56% as SE deployment increased from low to high. Project performance was measured by budget, schedule, and requirements satisfaction. This result is consistent with a previous analysis of the Return on Investment

(ROI) of software systems engineering [9], which concludes that there is a benefit of implementing SE practices up to an ideal degree, and that this degree increases with project size.

The deployment of SE leads to the definition of a process for system development. However, SE itself should be seen as an engineering discipline and not a process [10]. In literature and in industry, there is no global “systems engineering process” that is consensual and accepted by all [3, 7, 11]. The attempt to define one is also not a reasonable endeavour since, as stated in [10], this would constitute an attempt to develop a general problem-solving process. Instead, a “systems engineering process” may be defined and tailored for a specific type of problem, making use of concepts that are studied in the discipline. This will be the approach to SE adopted in this work.

We may, however, discuss some of the more consensual SE activities and their motivation, to paint a picture of what is SE. The first step in the development of a system is usually the proper definition of the problem to be solved, an activity that is based on requirements engineering. Requirements engineering consists of developing, documenting, and managing requirements. It is a continuous activity that focuses on deriving system requirements from stakeholder expectations, analysing requirements to derive additional requirements, maintaining traceability, and managing changes [12].

For the definition of a solution to be efficient, system complexity must be managed. The development of complex systems usually results in the decomposition of the system into multiple subsystems, with each being developed by a different team of specialists. To support this breakdown of the engineering effort, SE must enable a clear specification of the system architecture and a consistent definition of interfaces. With complexity being effectively managed, different solutions should be evaluated and compared to achieve one solution that meets the stakeholder needs as effectively and efficiently as possible. SE can assist this process by introducing methods for system analysis and trade-off analysis.

Verification and validation (V&V) activities are essential to make sure the correct system is produced and integrated correctly. The verification of a system relates to the evaluation of whether the defined requirements are met, while the validation of a system consists of determining if the system solution meets the stakeholder expectations [3]. It is important that the activities related to V&V start in the early phases of the life cycle (e.g., with the definition of testing procedures or validation criteria) to identify missing requirements, overlooked assumptions, or incorrect refinement of stakeholder expectations [13].

### **1.2.3 Motivation for Model-Based Systems Engineering**

Traditionally, SE has been implemented according to what modelling practitioners call a document-based approach [14, p. 15]. With document-based systems engineering (DBSE), the primary artifacts produced in the life cycle activities are a set of text documents, spreadsheets, diagrams, and presentations, e.g., concept of operations (ConOps) documents, requirements specifications, or Requirements Verification and Traceability Matrices (RVTM). Most of the effort of generating and maintaining these artifacts falls upon the systems engineers.

The artifacts produced with DBSE are a disjoint set of documents which are exchanged among clients, suppliers, and developers. With DBSE, the systems engineering effort ends up being centred

on the control of documentation, continuously guaranteeing that the documents are valid, complete, and consistent, and that the developed system complies with the documentation [14, p. 15]. The major problem with this approach to systems engineering is that it is more expensive than it needs to be [15]. The maintenance of the disjoint set of documents represents significant costs, and if it is avoided, documents become inconsistent and obsolete.

Model-Based Systems Engineering (MBSE), if practiced correctly, is the solution to these issues [15]. With this approach, systems engineers perform the same life cycle activities and produce the same set of deliverables, to enable the interaction with other engineering teams or stakeholders that work with a document-based approach. However, these deliverables are not the primary artifacts. With an MBSE approach, the primary artifact of the life cycle activities is a model of the system being developed, which integrates other discipline specific models and simulations [16] and evolves throughout the life cycle of the system. This model includes information relative to specification, design, analysis, and verification, consisting of elements that represent requirements, design elements, test cases, design rationale, and their interrelationships [14, p. 17]. All other artifacts are automatically generated from the system model.

The system model created with MBSE is developed using a modelling tool and contained in a model repository that centralizes information (see Figure 1.1). The model is essentially composed of a set of interconnected elements that can be reused across projects. The most common approach to communicate with stakeholders and other engineers is to create diagrams that show views of the system model, focusing on the aspects of the system that are relevant for a specific interaction. This approach is intended to replace document-based interaction, even though it often requires partial knowledge of a modelling language.

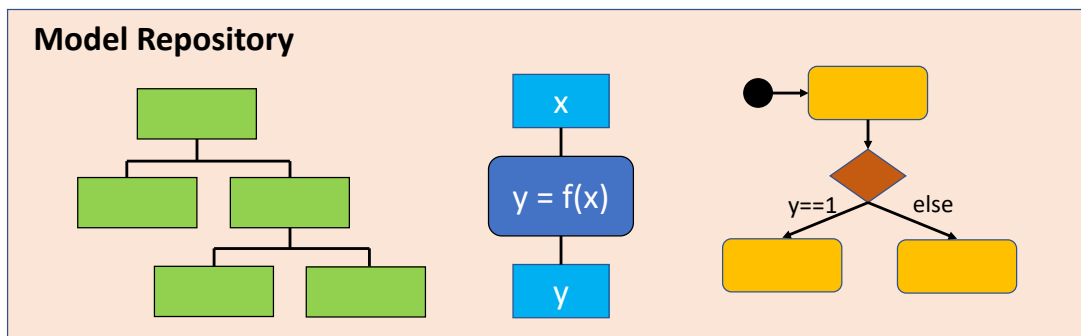


Figure 1.1: Example of elements of a system model contained inside a model repository.

MBSE aims to improve upon DBSE by merging modelling into current SE practices and by introducing a centralized management of the information across all domains, i.e., create a single source of truth [5]. The systems engineering community has a strong interest in MBSE, but empirical evidence supporting the benefits of MBSE is still lacking [17]. In fact, a poor implementation of MBSE is found to focus more on modelling than on SE and it generates unnecessary complexity [18]. For this reason, it can be a risk for organizations to transition from DBSE to MBSE. However, the same researchers that raise doubts regarding the adoption of MBSE, also believe that it can be beneficial if implemented correctly [17, 18].

Regarding the observed and perceived benefits of MBSE, the introduction of modelling into SE prac-



tices can provide enhanced communication among engineers and stakeholders, improving the shared understanding of the system. It can further improve the ability to manage complexity by providing multiple perspectives of the model [14]. The centralization of information provides the ability to evaluate consistency and correctness and to propagate changes made to the model to all the involved diagrams and artifacts. A better leveraging of Information Technology (IT) capabilities enables MBSE to ensure the reuse of existing models, automated generation of documents, easy impact analysis of design changes, and continuous requirements validation and design verification.

## **1.2.4 Deployment of Model Based Systems Engineering**

MBSE has been a research target ever since its mathematical foundation was introduced in 1993 [19]. In the last 15 years, the interest of researchers in MBSE has increased over time [20], and it was accompanied by a continuous improvement of the methods, languages, and tools.

The deployment of MBSE practices requires the mastering of what is defined in [15] as the three pillars of MBSE: a modelling language, a modelling method, and a modelling tool. The modelling language defines the kinds of elements that can be part of the model and the kinds of relationships that can be defined between these elements, providing the grammar that supports modelling. A modelling method provides guidance to the modelling effort, defining a set of tasks to be performed to create the model. Finally, the modelling tool, which complies with the rules of the modelling language, provides the necessary IT functions and the user interface for practicing MBSE. These are not simple diagramming tools, since they must be able to e.g., store and organize the system model, propagate changes, provide views of the model, and evaluate modelling correctness [15].

The advancement of MBSE practices depends on the maturity of each of the three pillars and of SE practices in general. One important enabler of MBSE is the OMG Systems Modelling Language (SysML) [3], an open-source language that was adopted by the Object Management Group (OMG) in 2006. The SysML standard has been continuously improved since it was adopted by OMG. In parallel, modelling tools have improved the support provided for the languages and the efficiency of the modelling process.

The evolution and adoption of MBSE practices relies on the availability of practical guidance and case studies, both as a way to challenge the methods and tools, but also to help new system engineers learn the trade. In this context, some important projects have been developed on the application of MBSE to space systems engineering. The Object-Oriented Systems Engineering Method (OOSEM) was applied to the development of a preferred system architecture for the FireSat [21], a hypothetical space system described in [22], while a different project focused on developing a CubeSat Reference Model [16]. The objective of this model was to be used by teams that are developing CubeSats as a starting point for their mission-specific models. This project resulted in practical examples regarding the modelling of behaviours [23] and technical performance measures [24], the development of an enterprise model [25], and the integration of the system model with simulation [26, 27].

The adoption of MBSE in an industry setting faces new challenges in comparison to research, e.g., the lack of engineers with the required skill set, the larger development teams, and the larger upfront

investment. However, the adoption of MBSE in the space industry has already started. In the European space community, different methods based on model-driven engineering are being developed, such as the Architecture Analysis & Design Integrated Approach (ARCADIA) and the ESA-led methodology Virtual Spacecraft Design (VSD) [5]. SysML has also been adopted by ESA, starting with the introduction of modelling practices in the ESA *Euclid* science mission [5]. This led to the development of an ESA specific SysML profile that is currently in an ESA SysML Toolbox, compiled together with a modelling methodology. ESA also developed a pilot study consisting of deploying MBSE to design the *e.Deorbit* mission, a project that involved three major industry players implementing three different methodologies [5]. In NASA, MBSE was also applied to real issues in the context of the MBSE Pathfinder effort in which four different teams addressed different focus areas using SysML [28].

MBSE may be a catalyst to the evolution of the engineering processes that are common today. Model-based approaches have been for many years the standard in other disciplines, such as electrical and mechanical engineering, and MBSE may be the key for obtaining “end-to-end support of engineering processes through models” [29]. This concept aligns itself with some trends in modern engineering such as “Digital Engineering” [29], “Industry 4.0” [29], and “Digital Twin” [30]. The integration of MBSE with concurrent engineering practices is also promising and is currently being developed in the Concurrent Design Facility (CDF) at ESA [31].

### 1.2.5 Integration of MBSE with Simulation Tools

The quantitative evaluation of performance is an essential part of the system design process. Complex system design often makes use of simulation models that are executed with the purpose of assessing system performance, via the post-processing of simulation results. The development and execution of simulation models is usually performed by domain specialist engineers. The tools that support this activity are often either domain-specific simulation tools or general-purpose simulation environments (e.g., MATLAB/Simulink or Dymola). This type of tools is also the preferred choice for the development of simulation models in the context of an MBSE approach. This is due to the following reasons:

- Systems modelling languages like SysML still do not adequately support complex mathematical modelling [32].
- Domain specialist engineers are already routinely using this type of tools for system analysis [33].
- Most organizations work with existing models developed with these tools, and the reuse of these models is essential for the reduction of project resource expenditure [34].

For these reasons, there is the need to integrate MBSE modelling tools with simulation tools. To ensure that this tool integration process respects the core tenets of MBSE, all the information relative to the system design specification must be centralized into a single source of truth, the system model. This means that mathematical modelling and simulation can and should be performed inside the scope of the simulation tool, but the system specification information used in the simulation model should be retrieved from the system model. In this work, we focus on the more general case of integrating MBSE

with a general-purpose simulation environment. There are two common approaches to achieve this type of integration: co-simulation and model transformation [35].

Co-simulation is the integration of various simulators to perform a composite simulation [36]. With a co-simulation approach, a communication channel is created between the MBSE modelling tool and an external simulation tool. The simulation process is managed by a simulation engine that is part of the MBSE modelling tool, because the system specification can be directly accessed by it. The external simulation tool is used for modelling and execution of mathematical models of higher complexity, either because it is better suited for the job or because it enables the reuse of previously developed models.

An approach based on model transformation consists of the automatic transformation of the system models contained inside the model repository into simulation models that can be executed in the simulation environment. This approach does not require the MBSE modelling tool to include a simulation engine, since the mathematical modelling and simulation can be performed completely inside the scope of a dedicated simulation environment. Instead, it requires the support of a standard that maps the semantics of the modelling language to the semantics used in the simulation environment. The system specification is exported from the MBSE modelling tool to the simulation environment and it is combined with the mathematical models to generate a functioning simulation model.

## 1.3 Literature Review

The implementation of MBSE both in industry and in research is not yet consistent. MBSE is still applied with “a set of different approaches which vary in scope and purpose” [37]. A taxonomy for MBSE has been proposed [37], defining three main categories: system specification repositories, system execution models, and design automation models. The first category consists of the use of formal models that are purely descriptive and static, while the second category focuses on the definition of parametric relationships between model elements, with the final purpose of using simulation to estimate system performance in a quantitative way. In this work we look at these categories as steps in the implementation of MBSE practices, starting with a static specification of the system, transitioning to executable system models, and finally achieving the development of design automation models. This third category can be seen as an extension to the system execution models, enabling simulation results to automatically change the system specification.

MBSE is a recent concept, and it is consequently still relatively unexplored from a research point of view [20]. However, this is not the only reason for the lack of maturity of MBSE, since there have been some challenges to the adoption of MBSE in industry. Research has been developed targeting the identification of these challenges, such as in [20, 38, 39]. From the identified challenges we selected those that are most relevant in the context of this work, which are:

- C1: Lack of practical guidance, case studies, and examples.
- C2: Lack of maturity of tool set and methods.
- C3: Tool integration limitations.

While the application of MBSE at the system-of-systems (SOS) level is a topic of interest in research [40, 41], it will not be included in this review since it is not relevant for the development of this thesis. The review of the most important existing related work will be divided into two sections, the first concerning the application of MBSE in support of space system design, and the second focusing on the specific issue of integrating simulation models with the system models produced with MBSE.

### 1.3.1 MBSE Deployment

The CubeSat Reference Model project mentioned earlier propelled relevant contributions related to this topic. The development of the CubeSat Reference Model, which follows the OOSEM, was first introduced in [42], followed by updates in the last years regarding the global status of the project [16, 43, 44]. These works target the lack of practical guidance in literature (C1), but the developed reference model is meant to be a reference for other projects and does not represent a real design process.

Some of the works in this topic consist of conceptual design processes, which focus mostly on the system-level of the architecture. MBSE has been implemented in the context of small satellite conceptual design, resulting in a decomposition of the system into subsystems and in the execution of mass and power rollups via the integration of a MBSE modelling tool with MATLAB [45]. This is however a very simplified case study, both in terms of modelling and system analysis. The MBSSE methodology [46] has been applied to the conceptual design of the Atmospheric Sample Return (ASR) mission, in the context of the CDF [47]. This work focuses on the identification of problems in the modelling process (targeting C2) and in the integration with the concurrent engineering practices, rather than obtaining a valid design for the system.

Some formal methods have also been proposed. Shortcomings associated with system models developed with object-oriented (OO) modelling languages (such as SysML or Capella) have been identified (e.g., lack of semantic accuracy) [48]. The use of an ontology description language is proposed instead. Formal methods have been integrated with MBSE to develop a framework for small satellite design [49]. The deployment of this formal languages is however very complex and its implementation in real system definition processes is yet to be demonstrated.

Some works concern the application of MBSE in lower levels of the system architecture, specifically in support of subsystem design. A satellite communication system architecting framework has been developed with MBSE [50]. The implemented design scenario is quite simplified, and the subsystem is not modelled inside the scope of a complete spacecraft model.

Most of the existing work on the application of MBSE in the design of spacecraft subsystems concerns the design of attitude systems [51–54]. The process of deriving subsystem requirements of the ADCS from customer needs was developed using SysML [51]. Requirement verification using simulation was also performed by reusing an ADCS simulation model. A development process for AOCS software integrated with code generation has been proposed, using SysML [52]. The reliability analysis of an ADCS using simulation was developed using a model-based method named RAMSAS [55]. These works focus on specific aspects of the subsystem design process and are not developed in the context

of a spacecraft design process. The design of an ADCS has been developed with the Vitech MBSE methodology [54]. However, the presented design process is not complete (excluding e.g., simulation or requirements verification) and the ADCS is modelled outside the context of a spacecraft or a mission.

There is a lack of case studies that present the design process of a subsystem using MBSE (consistent with challenge C1), and subsystem design activities implemented with MBSE are usually presented outside the context of a complete spacecraft design. These practical examples are important because MBSE is aimed to support the complete design of systems, allowing different teams to work concurrently on the design of different subsystems. This type of case study would help to validate MBSE practices.

### 1.3.2 Integration of MBSE with Simulation Tools

This topic is tightly related to the adoption challenge C3 (tool integration limitations). There are two common approaches to integrate MBSE with simulation: co-simulation and model transformation [35]. Currently, the most mature approach is co-simulation, which has been shown in literature supporting space system design. Co-simulation was used to enable the reuse of a previously developed ADCS simulator in [51]. In the context of the CubeSat Reference Model, a co-simulation approach was developed to integrate SysML with MATLAB scripts and other more complex simulation workflows which integrate multiple simulation tools [27]. With these co-simulation approaches, changes made in the system model architecture do not propagate automatically to the simulation models. Also, these approaches are not scalable for more complex projects [35].

A different approach was proposed in which SysML modelling patterns are created to help specify an *analysis context* that can be executed [56]. Integrating parametric modelling with the semantics of behaviour specification, which are more directed for execution, enables execution to be performed solely by the MBSE tool execution engines. However, these execution engines are not prepared to deal with complex mathematical models, and so a similar co-simulation framework will also be required.

Model transformation approaches are much less mature, mostly because of the lack of standard maturity and tool support. Since the application of this approach in space system design is rare, here we will also consider other types of engineering systems. This type of approach has been applied to the simulation of a simple electrical circuit, using the SysML2Simulink standard [32]. A model transformation based on triple graph grammar has been applied to the simulation of the discrete/continuous behaviour of an inverted pendulum system, integrating SysML with Stateflow and Simscape [57]. A process for deriving executable simulation models from SysML using QVT has also been developed [58]. The QVT language [59] enables the formal definition of bidirectional model transformations. These examples demonstrate that model transformations ensure centralization of information in the system model, which remains automatically consistent with the simulation models. However, the available examples concern only very simple systems.

## 1.4 Research Contributions and Thesis Outline

In this thesis we apply an MBSE methodology in support of a design process which targets a subset of a complete space system design. This process concerns the preliminary design of a spacecraft subsystem. The system design definition requires the support of the activity of “system modelling and simulation” as defined by the ECSS in [60]. This standard defines that preliminary design activities are supported by the use of a Functional Engineering Simulator (FES) to perform system functional design validation. The development of this simulator is part of this work, and it is the result of the integration between the MBSE modelling tool and a simulation environment. The design process itself and the structure of this simulator are formally defined in Chapter 2.

In this work, the first step is to select an adequate MBSE methodology, to provide a base for the work developed. We must simultaneously select a modelling language, a modelling tool, a simulation environment, and a method for integrating MBSE with the simulation environment, since all of these are linked. This selection process and an overview of the various enabling methods, tools, and languages are presented in Chapter 2.

The proposed MBSE methodology results from the adaptation and extension of an off-the-shelf methodology which is tailored to the specific design process targeted in this work. A description of the developed methodology is presented in Chapter 3. The methodology is then applied to the design of an ADCS of a small satellite.

The context for the ADCS design scenario, as part of the overall definition of system and mission, is defined as a set of inputs to the design process, which are outlined in Chapter 4.

Also presented in this chapter is the step-by-step application of the methodology to this particular design scenario. This presentation includes all the details of the ADCS design, which are used to illustrate the implementation of the methodology. While these will certainly convey the design process of the ADCS to the reader, the purpose of this work is to present an application of the methodology generically. The application of the methods developed for the integration of MBSE with simulation is also presented at the end of the chapter, including a description of the simulation models used and a summary of the simulation results obtained.

The application of MBSE in a design process of a spacecraft subsystem, inside the scope of a spacecraft design process, constitutes a case study that is not currently available in literature. This is a contribution that addresses the challenge C1 (lack of practical guidance). We also identify limitations of the methods and tools involved, addressing the challenge C2 (lack of maturity of tool set and methods). Finally, we address the challenge C3 (tool integration limitations), by demonstrating how an MBSE tool can be integrated with a simulation environment in a way that respects the core tenets of MBSE, harnesses all the benefits of MBSE, and is consistent with the skill set of domain specialist engineers. To this end, a MATLAB program is developed to extend the capabilities of the MBSE tool in the process of transforming SysML specification into a functioning simulator.

## Chapter 2

# Theoretical Background

### 2.1 MBSE Methodology Review

In this section we discuss the selection of an MBSE methodology that can be used as a base for the work developed in this thesis. The selected methodology will then be tailored to the specific design process implemented in this work (see Chapter 3). When using terms whose meaning is specific to a certain MBSE methodology, these are shown in *italic* (e.g., *analysis context*). Before different MBSE methodologies can be reviewed, we first present some definitions derived from [61]:

- A **Process** is a sequence of tasks performed to achieve a specific goal. It specifies what is to be done, without defining how it must be done.
- A **Method** specifies how a task must be performed and it is composed of a set of techniques required to perform it.
- A **Tool** is an instrument that supports a specific method to improve the efficiency of a task.
- A **Methodology** is composed of related processes, methods, and tools. It is designed to be applied to a certain class of problems.

A review of some of the more notable MBSE methodologies has been done in [62], and it is still a reference for more recent works [45, 50]. The methodologies considered in this work are, from those discussed in [62], the ones applicable to the development of general (or non-specific) systems. An evaluation of whether each of the methodologies in [62] meets this requirement was done in [50]. Three possible methodologies emerge:

**INCOSE Object-Oriented Systems Engineering Method (OOSEM)** [14] It is a top-down, scenario-driven approach. It combines object-oriented concepts with more traditional systems engineering methods, and it is consistent with a recursive “Vee” life cycle process [63]. This methodology defines a detailed process for the definition of the SOI, starting with an analysis of stakeholder expectations, followed by the specification of the system as a black box, and finishing with the definition of the system architecture in two stages: *logical architecture definition* and *candidate physical architecture synthesis*.

**IBM Telelogic Harmony-SE** [64] It uses a service request-driven modelling approach that is consistent with the “Vee” life cycle process. Functional decomposition is accomplished through decomposition of *activity operational contracts*, which describe *state/mode changes* or *operations*. The three top-level process elements are: *requirement analysis*, *system functional analysis*, and *architectural design*. This methodology is specifically suited for initial system architecture design [50].

**Dori Object-Process Methodology (OPM)** [65] It combines formal visual models with natural language sentences. Every formal model is specified by a semantically equivalent sentence, expressed using a customized dual-purpose language (i.e., both oriented towards humans and machines). This methodology regards everything in the universe as one of three elements: an *object*, a thing that exists or has the potential to exist; a *process*, a pattern of transformation that an object undergoes; or a *state*, a situation an object can be at [62]. The Dori OPM is based on a holistic systems paradigm, supporting very rich modelling semantics that can be used to model all the aspects of a system in one single view [50]. The system development process is divided into the following sequential steps: *requirement specifying*, *analysing and designing*, *implementing*, and *using and maintaining*.

A comparison of the key aspects of the methodologies is shown in Table 2.1. The OOSEM and Harmony-SE are supported by a widely used modelling language — the SysML — while the OPM uses its own customized modelling language. For this reason and because the OPM is also based on holistic semantics, it is more difficult for less experienced system architects to learn and correctly implement this methodology. In terms of tool support, only the Dori OPM has a dedicated framework tool. The Harmony-SE and OOSEM are instead tool- and vendor-neutral. The OOSEM emphasizes too much on a specific type of system, namely Cyber-Physical Systems (CPS), but the detail in the definition of the OOSEM framework compensates for this issue when compared to Harmony-SE.

| Methodology           | OOSEM               | Harmony-SE   | OPM            |
|-----------------------|---------------------|--------------|----------------|
| Emphasis              | software & hardware | general      | general        |
| Detailed Framework?   | Yes                 | No           | No             |
| Widely used language? | Yes                 | Yes          | No             |
| Holistic Semantics ?  | No                  | No           | Yes            |
| Tool Support          | Tool-neutral        | Tool-neutral | Dedicated tool |

Table 2.1: Comparison of the reviewed methodologies’ key aspects

Being based on OO concepts, the OOSEM may induce system architects into considering specific objects as isolated even when that is not possible [18]. However, the methodology is well documented [14], there is an extensive example on its application in the context of space systems architecting [21], and many of the reviewed works use this methodology [42, 45, 50, 56].

For the reasons above, we selected the OOSEM methodology, and consequently SysML as the modelling language. This selection process considered the compatibility with the selected tools and model transformation standard, which are discussed later in this chapter.



## 2.2 SysML

The Systems Modelling Language (SysML) [66] is a “general-purpose graphical modelling language that supports the analysis, specification, design, verification, and validation of complex systems” [14, p. 29]. It has grammar and vocabulary like any natural language (e.g., English, Portuguese) but its vocabulary is composed of graphical notations. MBSE requires a precise and unambiguous language to fully achieve its purpose. Natural languages are usually ambiguous, containing a lot of words that are synonyms or have multiple meanings, which is in general not the case for SysML. Additionally, MBSE requires a language that is graphical, so that elements and their relationships are easy to visualize and understand.

SysML is a standard from the Object Management Group (OMG) that reuses a part of the UML 2 standard (widely used in software engineering), extending it to include elements that are necessary for the application of SE to general systems development. SysML leverages the OMG XML Metadata Interchange (XMI) standard, which enables the exchange of model data across different tools from different tool providers. The SysML version used is 1.6, published in December 2019.

The complete specification of SysML is quite detailed and extensive (available in [66]). Thus, we focus this overview of SysML on the general diagram notation and taxonomy and on some of the most used elements of the language. When a word or expression refers to a SysML concept, having a distinct meaning in this context, it is shown in *italic*, e.g., *value property*. Additional clarification regarding SysML nomenclature will be provided when a diagram contains elements that are not presented in this section. A detailed review of SysML notation can be found in [14, 15].

### 2.2.1 Diagram Notation and Taxonomy

SysML diagrams show views of the underlying system model. These are composed by a diagram frame and a header, and the latter is located on the top of the diagram and contained in a rectangle with a corner cut off (see Figure 2.2). The header describes the diagram kind, the type and name of a model element, and the name of the diagram. The model element mentioned in the header can be viewed as the element represented by the diagram frame. Inside the diagram frame a set of elements are shown, represented as nodes of different shapes or as lines and arrows connecting two different nodes, which represent relationships between elements. The elements presented in the diagram represent what is specified in the system model but reflect only a part of the specification.

SysML includes nine different kinds of diagrams, each with a different function. In this work, the diagram kind abbreviations are shown in bold (e.g., **bdd**). The diagram kinds are summarized below (and the associated abbreviations are shown in parentheses):

- **Package Diagram (pkg)**: It is used to represent the organization of a model in terms of how model elements are contained in different *packages* and how the different *packages* are organized according to a specific hierarchy.
- **Block Definition Diagram (bdd)**: It represents elements of definition and the structural relationships between these elements, such as *composition* and *classification*. Elements of definition

serve as types for the model elements represented in other kinds of diagrams,

- **Internal Body Diagram (ibd)**: It specifies the internal structure of a block. Usages of blocks are displayed in the form of *part and reference properties*, and a set of connections between these are represented. This diagram represents how the parts are assembled together to create a valid instance of a block and how an instance of the block should be connected to external entities.
- **Parametric Diagram (par)**: It displays the internal structure of a block focusing on *value properties* and their connections to *constraint parameters*. This diagram provides support to engineering analysis by enabling the specification of mathematical relationships between *value properties*.
- **Use Case Diagram (uc)**: It represents a set of use cases — “the externally visible services that a system provides” [15, p. 77] — and the *actors* that invoke or participate in them. This diagram is a black box representation of system function used to analyse and document stakeholder needs.
- **Activity Diagram (act)**: It is a kind of behaviour diagram. It always represents an *activity*, which is used to specify behaviour that, through a controlled sequence of actions, transforms inputs into outputs. This flow-based representation of behaviour is appropriate for the representation of algorithms and complex control logic.
- **Sequence Diagram (sd)**: It is a kind of behaviour diagram. It always represents an *interaction*, which specifies behaviour in terms of a sequence of message exchanges between structural elements. This service-based representation of behaviour is appropriate for the representation of how and in what order elements invoke services and send signals to each other. These are appropriate for software behavioural modelling or test case specification.
- **State Machine Diagram (stm)**: It is a kind of behaviour diagram. It always represents a state machine, which specifies behaviour in terms of a block’s states and the transitions between them. This state-based representation of behaviour is appropriate for the representation of a classifier behaviour (the main behaviour of the block that starts at the beginning of its lifetime and generally finishes at the end of it [14, p. 128]). It is specifically suited for blocks which have a set of different states relevant to their behaviour specification.
- **Requirement Diagram (req)**: It represents text-based requirements and their relationships with other model elements, including the relationships between requirements.

## 2.2.2 Basic elements of the language

The block is the basic unit of structure in SysML [15, p. 26] and it is used to model any type of entity. A block does not represent an instance of an entity, but rather a set of instances that share the same definition. Instances of blocks can then be created to represent a specific use of the modelled entity. As described in [14, p. 97], a block can model a physical entity, software, hardware, a data component, a person, a facility, or an entity that flows through a system. As an example, a block may be created to represent a wheel of a car, which can be named *Wheel*. Four instances of *Wheel* are used in the model

of the car, one for each quarter of the vehicle, but not all of these behave equally (e.g., front wheels are steered while rear wheels are not).

The representation of a block is presented in Figure 2.1(a). Blocks have many different features, such as *part properties*, *value properties*, and *ports*. *Part properties* specify the decomposition of a block into elements, *value properties* define quantifiable attributes of a block, and *ports* specify interaction points with the exterior (both to provide or request services and to exchange matter, energy, or data). Figure 2.1(a) shows the various types of features in different compartments, except for the *ports* that are represented as green-filled squares on the boundaries of the block. Using the same example from above, the element *Wheel* may own *part properties* typed by blocks called *Rim* and *Tire*, a *value property* named *radius*, and a *port* representing the transmission of torque or power to the wheel.

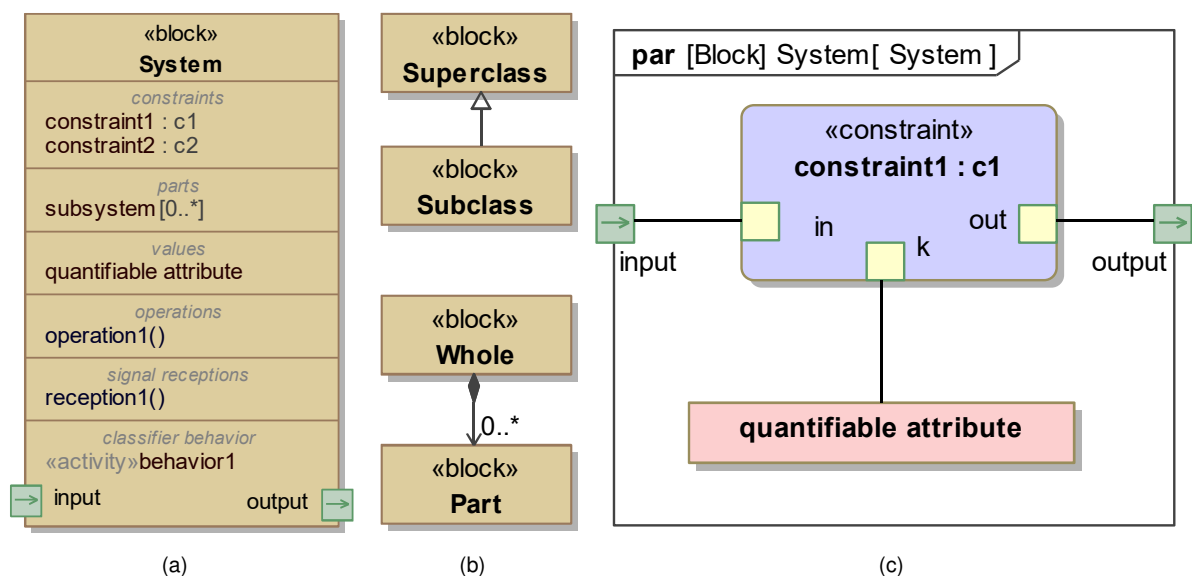


Figure 2.1: (a) Representation of a generic block and its features. (b) Representation of a *generalization* relationship, on the top, and a *composite association*, on the bottom. (c) Parametric representation of the internal structure of a block

Use cases specify the behaviour of the system, but in terms of what is expected from stakeholders. Excluding these, the behaviour of a system can be modelled with *activities*, *interactions*, and state machines, each being tailored for a specific behavioural modelling need. Behaviour can be allocated to structure by defining one of three types of behavioural features: classifier behaviour, *operations*, and *receptions* (see Figure 2.1(a)). A classifier behaviour is usually specified using a state machine, even though sometimes *activities* are also used. An *operation* is a synchronous behaviour that is invoked when a client calls it. Parameters may be passed in with the service request and also passed out to the client when the request has been handled. A *reception* is an asynchronous behaviour that is requested with a signal (a signal specifies a message with a set of attributes). The details regarding the invocation and request of *operations* and *receptions* may be specified with an *interaction*, and the behaviour that they represent may be specified with an *activity*.

There are two types of relationships between blocks that are especially relevant in SysML: *Composite*

*associations* and *generalizations*. *Composite associations* specify that a block, the whole, is composed by another block, the part (see Figure 2.1(b)). It also defines what is an acceptable range of multiplicity for the part block (in Figure 2.1(b) the whole may be composed by any number of parts, including none, which is specified by a multiplicity represented as “0..\*”). The *composite association* and the *part property* feature of a block represent the same relationship. The *generalization* defines that a block is a subtype of another block, called the *superclass* (see Figure 2.1(b)). If nothing is specified against it, the features of the *superclass* are passed to the *subclass*. For example, a block called *Front Wheel* is a subtype of *Wheel*.

In SysML, equations and inequalities can be modelled with *constraint blocks*. This is a special type of block that defines a mathematical expression which involves a set of *constraint parameters*. Blocks may have *constraint properties*, a type of feature that attributes a constraint expression to the block, imposing a specific mathematical relationship on its *value properties*. *Constraint properties* are usually typed by *constraint blocks*. A **par** diagram is then used to show how one or more *constraint properties* are connected to the block’s *value properties* and *ports* (see Figure 2.1(c)).

SysML enables the definition of text requirements and defines a set of relationships that these can have with other types of elements and between each other. These relationships are usually represented with **req** diagrams, as the one in Figure 2.2. A requirement may contain other requirements, and this containment relationship is represented with a crosshair notation (which is also valid for *packages*). Requirements can also be derived from other higher-level requirements, and this is specified with a *derive requirement* relationship. A requirement may be refined by another more concrete requirement or by a *constraint block*, which provides an exact mathematical expression to represent the requirement, enabling automated verification to be performed. The requirements can also be satisfied by the attributes of blocks and verified by *test cases*, which are usually specified with *interactions*.

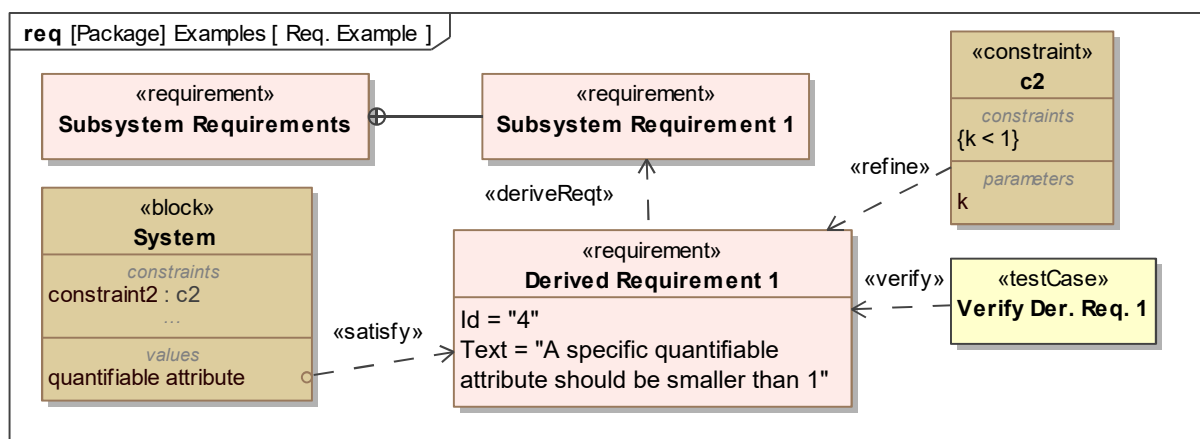


Figure 2.2: Representation of the most common relationships involving requirements.

*Allocations* are a type of cross-cutting relationship which has a few different possible uses. Behaviours can be allocated to structural elements, as shown in Figure 2.1(a), software can be allocated to hardware, and requirements can be allocated to structural elements with a *satisfy* relationship, as shown in Figure 2.2. *Allocations* introduce traceability between different types of elements.

## 2.3 Object-Oriented Systems Engineering Method (OOSEM)

The Object-Oriented Systems Engineering Method (OOSEM) is “a top-down, scenario-driven process that uses SysML to support the analysis, specification, design, and verification of systems” [14, p. 397]. It is a hybrid approach that combines modelling techniques and a solid systems engineering foundation with object-oriented concepts (e.g., blocks, encapsulation, and inheritance).

In this work, we implement the OOSEM as described in [14], where it is defined as a part of a higher-level process. This higher-level process addresses the complete life cycle of the system definition, and it is applied recursively to all levels of the system hierarchy, being consistent with the “Vee” development process (see Figure 2.3). The life cycle is defined as “the evolution with time of a SOI from conception to retirement” [3]. This higher-level process includes four different sub-processes, as shown in Figure 2.3, which are defined in [14] as follows:

- The *Manage System Development* process consists of project planning and control, including all the SE management activities. This is a process that starts in the beginning of the life cycle and ends with the disposal of the system.
- The *Specify and Design System* process starts with the analysis of system requirements and definition of the system architecture. Then, requirements are derived and allocated to successively lower-levels of the architecture, while the design definition at these levels is developed. This process is the one implemented by OOSEM.
- The *Develop Hardware, Software, Database, and Operational Procedures* process concerns the design, implementation, and testing of the lower-level elements of the system.
- The *Integrate and Verify System* process consists of the successive integration and test of the elements of the system, from the lower-levels of the architecture up to the system-level.

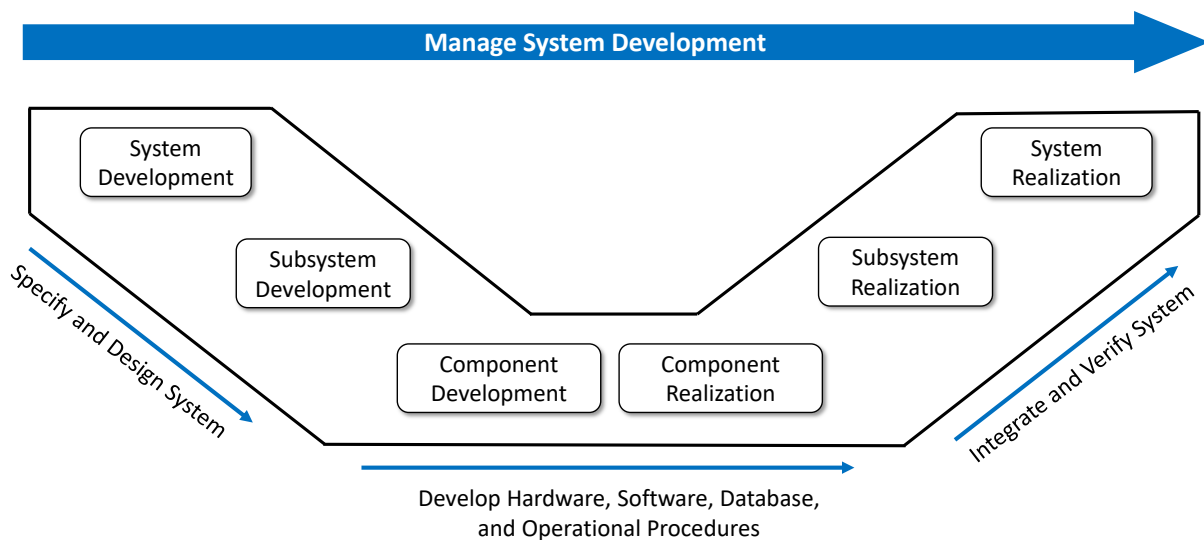


Figure 2.3: “Vee” model of the SOI life cycle.

The *Specify and Design System* process is the focus of this work, and it can be divided into separate activities as shown in Figure 2.4. The *Analyse Stakeholder Needs* activity is only applied at the mission-level, while all the other activities are applied recursively at each level of the system hierarchy. A description of each of these activities follows:

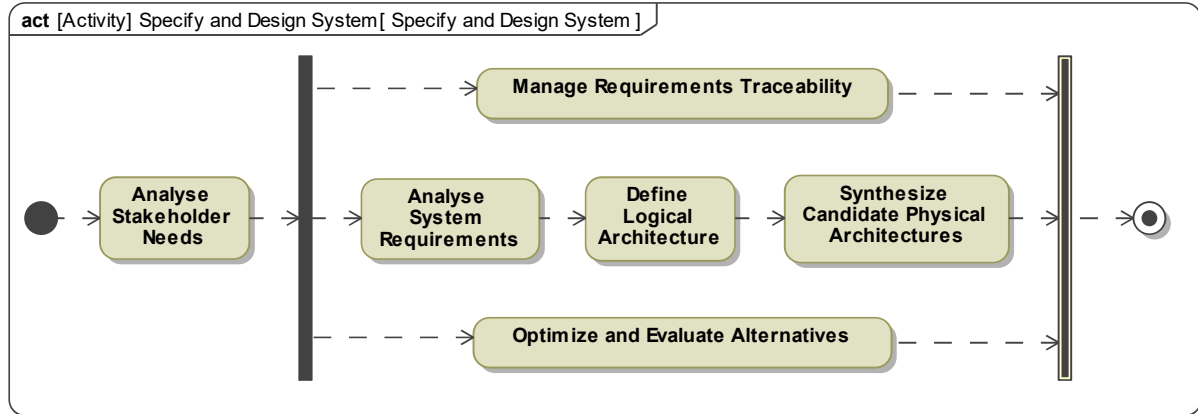


Figure 2.4: OOSEM process (each node represents an activity).

**Analyse Stakeholder Needs** This activity focuses on the elicitation of stakeholder expectations and derivation of mission-level requirements. When an “as-is” system configuration exists, it is characterized, and causal analysis is performed to identify potential improvement areas. The mission requirements are specified, which includes: (i) the definition of the enterprise use cases, to represent each mission objective; (ii) the definition of a domain model, to establish the context of the system and enterprise; and (iii) the capture of quantitative attributes that reflect mission-level performance requirements.

**Analyse System Requirements** The purpose of this activity is to characterize the system (or system element) in terms of its black box representation. First, for each of the use cases defined before, all the relevant scenarios are modelled with *activities* and *interactions*. Then, the system context is defined with an **ibd** and the critical system properties and constraints are captured. Finally, the system black box representation, requirements, and state machine behaviour are specified.

**Define Logical Architecture** An intermediate level of abstraction between the black box representation and the physical architecture is specified. It results in the definition of the logical decomposition of the system as well as its internal structure. The way the logical components interact to realize the previously modelled behaviours is defined, and a black box specification of the components is created.

**Synthesize Candidate Physical Architectures** Alternative physical architectures are synthesized and defined in terms of physical components distributed across nodes and relationships between these components. The physical components may include a combination of hardware, software, data, and operational procedures. For this reason, software, hardware, and data architectures must also be specified. The physical components’ black box representation is also defined.

**Optimize and Evaluate Alternatives** This activity is invoked throughout the complete OOSEM process, and it includes: the identification of which analyses are required, the specification of an *analysis context* for each of these, the use of **par** diagrams to model constraints, and the execution of engineering analysis. This activity requires some adaptation as is discussed in Chapter 3.

**Manage Requirements Traceability** This activity is also invoked throughout the complete OOSEM process, and it builds upon the definition of a *specification tree*, which defines an organizational scheme for the capture of requirements at each level of the system hierarchy. After the text-based requirements are captured, relationships are defined between them and other elements. A continuous analysis of traceability gaps and management of requirement updates must also be ensured by systems engineers.

## 2.4 Tool Selection and Integration

The development of this work requires the selection of an MBSE modelling tool, which supports SysML, and a general-purpose simulation environment, which will be used to create and execute simulation models. Regarding the modelling tool, the Cameo Systems Modeler (CSM) from NoMagic was selected. This was selected because modelling tools from NoMagic are the most commonly used in literature (e.g., [42, 45, 54, 56]) and the CSM provides all the required functions for this work, including support of the SysPhS standard, which is discussed later in this section. MATLAB/Simulink was selected as the simulation environment due to previous experience and because it is supported by the SysPhS standard.

As mentioned in Chapter 1, under an MBSE approach, simulation should be performed by integrating the MBSE modelling tool with a dedicated simulation tool, which in this case is MATLAB/Simulink. With this in mind, part of this work is dedicated to the development of a framework that enables the generation of functional simulation models from SysML specification. This framework is described in Section 3.1.3. Its purpose is to generate a complete Functional Engineering Simulator (FES), as defined in the ECSS [60]. The structure of this type of simulator is defined in [60] and shown in Figure 2.5.

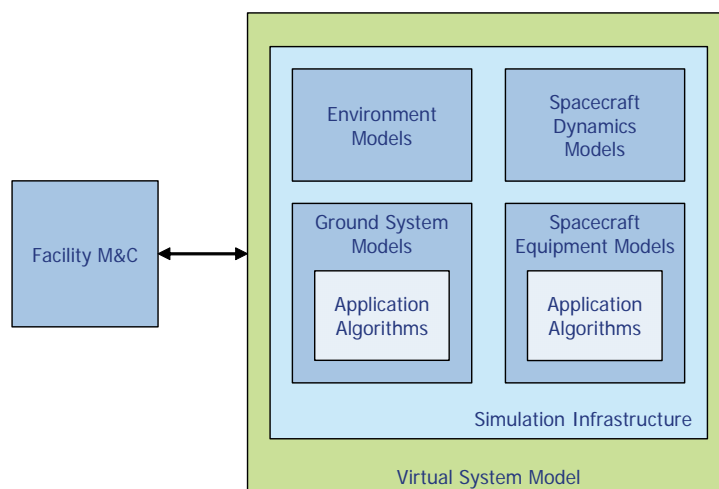


Figure 2.5: Functional Engineering Simulator (FES) structure (Figure 6 in [60]).

This simulator is used in support of a preliminary design process, being executed to determine the values of a set of technical performance measures (TPM), for a specific system configuration. The TPMs measure quantitative attributes of a system element. These are derived from Measures of Performance (MOP), which concern the performance of the system as a whole. The MOPs in turn are derived from Measures of Effectiveness (MOE), which reflect attributes of the mission in the perspective of the stakeholders. This process ensures traceability between high-level mission attributes and lower-level system element attributes. The use of the terms TPM, MOP, and MOE and their definition is not consensual in literature. The terms and definitions above, consistent with [24], will be the ones used in this work.

The framework mentioned above is based on a model transformation approach. This type of approach is described in more detail in Section 3.1.3 and in Figure 3.7. It is more appropriate than a co-simulation approach for this work as well as larger-scale applications since:

- More information is centralized in the system model, including simulation parameters and results.
- Domain specialists may develop and execute simulation models using tools they already master.
- The simulation model is automatically consistent with the system specification contained in the MBSE system model.
- The process of model transformation is not too complex and is not hard to implement or maintain.

The selection of this type of approach is dependent however on the existence of a supporting standard that is sufficiently mature. The QVT might be the best option for defining model transformations that are not completely tailored for a specific application, but there is a lack of practical guidance for the use of this language, and there is also lack of support from the most common MBSE modelling tools. We selected instead a more recent OMG standard, the SysML Physical Interaction and Signal Flow Simulation (SysPhS). It concerns the integration of SysML with Simulink, Simscape, and Modelica, defining a mapping between SysML elements and platform-specific simulation elements. This standard is extensively documented [67] and is supported by commercial MBSE tools, such as the CSM.

The most relevant part of the element mapping between SysML and Simulink defined by this standard is presented in Table 2.2. The CSM implements the SysPhS standard with the *Simulink Export* function, which transforms any block specified in SysML into a Simulink-based simulation model.

| <b>SysML</b>                                        | <b>Simulink/Stateflow</b> |
|-----------------------------------------------------|---------------------------|
| Block                                               | Subsystem Block           |
| Block with a state machine as classifier behaviour  | Stateflow Block           |
| <i>Constraint block, typing constraint property</i> | S-function                |
| <i>Port</i>                                         | Input/Output block        |
| Connector                                           | Line                      |

Table 2.2: Mapping between SysML and Simulink/Stateflow constructs specified by the OMG SysPhS.

The SysPhS standard provides comprehensive support to the transformation of SysML structural modelling, namely the blocks, their *ports*, and the connections between blocks. The SysML parametric modelling, specified using **par** diagrams, is also supported, and is mapped to S-functions in Simulink.



However, this standard lacks support for the transformation of behavioural modelling (i.e., the specification of how the blocks function), which is essential to specify simulation models.

As mentioned before, in SysML the behaviour is modelled with: (i) *activities* and *interactions*, which are not supported by this standard; and (ii) state machines, which are supported when defined as the classifier behaviour of a block, being mapped to Stateflow semantics. This lack of support, which results from lack of maturity of the available standards and the SysML language, poses an obstacle to the implementation of a model transformation framework. However, in a first approach to the development of a model transformation, and in the context of preliminary design, this issue may be surpassed.

The CSM also implements two *stereotypes* defined by the SysPhS standard, the *«SimulinkBlock»* and the *«SimulinkParameter»*. SysML *stereotypes* are used to extend an existing metaclass to create a new or modified concept. The first *stereotype* is applied to a block and the second to a *value property* of that same block, with the purpose of defining blocks in the SysML model that are mapped to platform-specific simulation constructs (i.e., to enable the import of simulation constructs to SysML).

## 2.5 Design Process Definition

The life cycle management is usually an essential process for the definition of the system. In this context, graphical representations of the system life cycle are often used, some of which are linear and sequential. However, iteration and recursion are essential to meet stakeholder needs effectively and efficiently. Iteration can be defined as the repeated application and interaction of processes at a given level in a system hierarchy. [3, p. 32] Recursion is instead the repeated application and interaction of processes at successive levels in the system hierarchy [3, p. 32].

Various life cycle models exist (e.g., Waterfall [68], Spiral [69], Vee [63]), and the selection of a model is usually closely related to the system definition methodology used. The methodology used in this work is consistent with the “Vee” life cycle model, which is shown in Figures 2.3 and 2.6. Using this model as a reference, we define in this section the design process to which the proposed methodology will be applied, as well as its context and scope (see Figure 2.6).

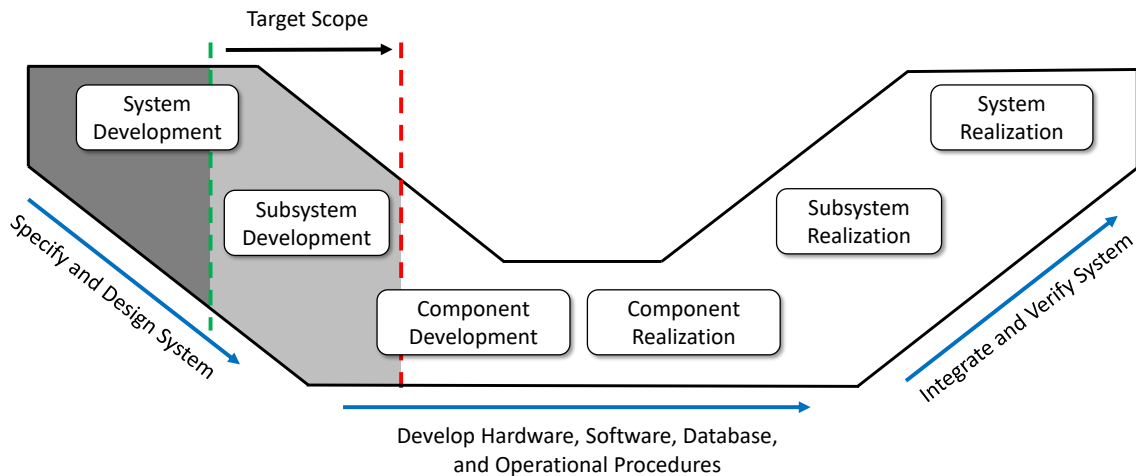


Figure 2.6: Definition of target scope using the “Vee” model.

The design process considered in this work targets the subsystem preliminary design phase. Thus, it is assumed that previous work has been developed in previous life cycle stages, and that the OOSEM has been applied at the system-level before it can be recursively applied at the subsystem-level. Considered to be subsequent work is: (i) the detailed design of the subsystem ; (ii) component development, procurement, and realization; and (iii) assembly, integration, and testing (AIT). In addition, this work will also not include tasks related to the preparation of V&V activities, such as the development of verification plans and definition of validation criteria, which should be performed as part of the specification and design of the system. This is because these tasks require a more complete definition of the system and also because it is not possible to validate them within the scope of this work.

In this work, four architecture levels are defined, as shown in Figure 2.7. A mission is usually composed of more than one system, including at least one spacecraft and one ground station. The system that is most relevant in this work is the spacecraft. It is decomposed into several subsystems (one of which is the ADCS), and each subsystem is decomposed into components. These can be hardware components (e.g., sensors and actuators) or software components (e.g., estimators and controllers).

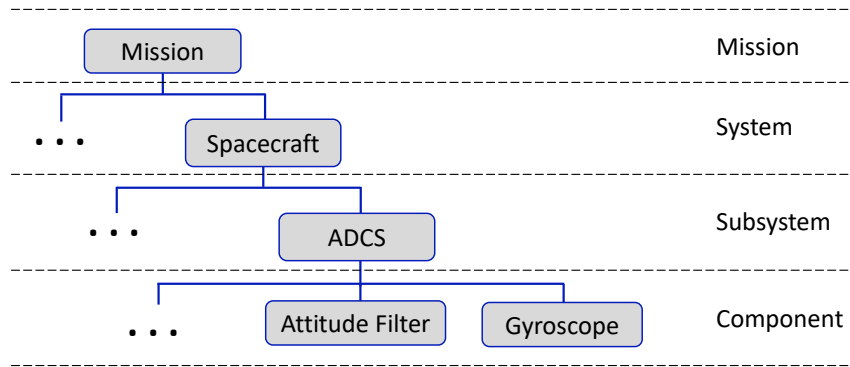


Figure 2.7: Architecture levels defined in this work.

Since this work concerns a subsystem design process that is part of a broader system design process, it is important to clearly define the scope and context of the former as part of the latter. Because of that, we specify the subsystem design process as a black box, i.e., in terms of inputs, outputs, and performed tasks, as it is common in the systems engineering discipline. The process, as shown in Figure 2.8, includes the tasks that are part of three different processes defined in the ISO/IEC/IEEE 15288: Architecture Definition, Design Definition, and System Analysis. The inputs and outputs are defined in accordance with the definition of the OOSEM process presented in [14].

Some of the inputs identified in Figure 2.8 concern the mission- and system-levels. The mission context specifies the different entities that are involved in the mission as well as the interaction of the system with the external environment, which is relevant to the design of most subsystems. The system white-box specification provides the context for the subsystem development. This consists of the decomposition of the system into subsystems and the definition of interactions between subsystems. Also relevant to the design of each subsystem is the system behaviour specification, since this behaviour will be divided into simpler behaviours that are allocated to each subsystem.

The subsystem-level inputs specify what is required for a specific design configuration to be valid.

The specification of the subsystem is defined with a black box representation, which includes the identification of relevant quantitative properties and constraints, the definition of the subsystem interface, and the identification of services that it must provide. The other input consists of a complete list of subsystem requirements, which also includes the design constraints defined in previous phases of the project. The definition of these requirements must include the status of the verification process and the traceability with higher-level requirements, so that requirement change impact analysis may be performed.

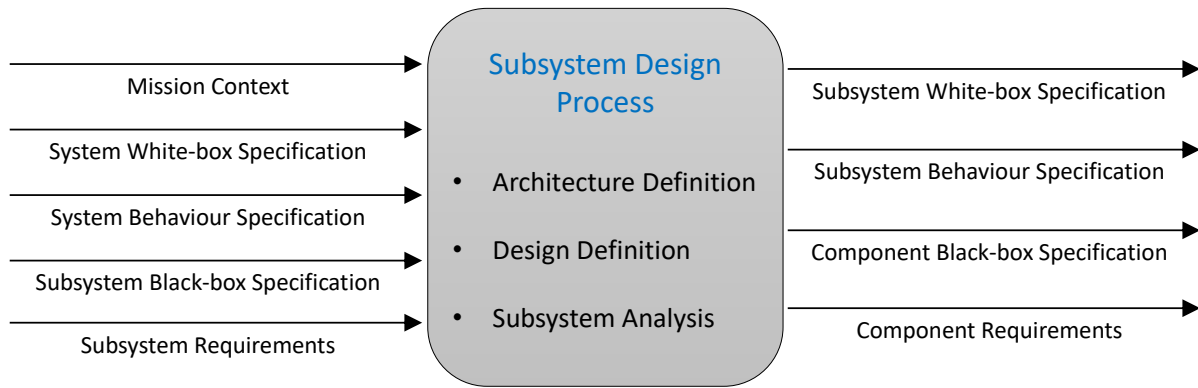


Figure 2.8: Black-box representation of the subsystem design process.

The outputs of the design process are similar to the inputs since this design process is part of a recursive application of the OOSEM at successive levels of the architecture. The design process results in the specification of the subsystem white box representation and its behaviour. Parts of this behaviour are allocated to each of the components. As component-level outputs, a black box representation of each of the components is specified together with a set of requirements, which are compiled in a complete list of component requirements.

## 2.6 ADCS Design and technology

Since the methodology developed in this work is applied to the design of an ADCS, we must review some fundamental concepts of ADCS design and technology. The purpose of an ADCS is to determine and control a space vehicle's attitude, i.e., its three-dimensional orientation with respect to a specified reference frame [22, p. 565]. The concepts reviewed in this section will be centred on the design of an ADCS based on active control techniques.

An ADCS that implements this type of control technique has a structure as represented in Figure 2.9, which is consistent with the control system structure defined by the ECSS [70]. The ADCS is composed of sensors, which draw measurements that are used by an attitude determination algorithm to derive an estimated state. The attitude guidance translates a reference state defined by the mission into a desired state which, together with the estimated state, is used by attitude control algorithms to derive control commands. The actuators receive these commands and actuate on the attitude of the spacecraft.

According to [22], the first step in the design of an attitude system is the definition of the ADCS control modes and the derivation of requirements for each mode. Some typical modes are [22]: (i)

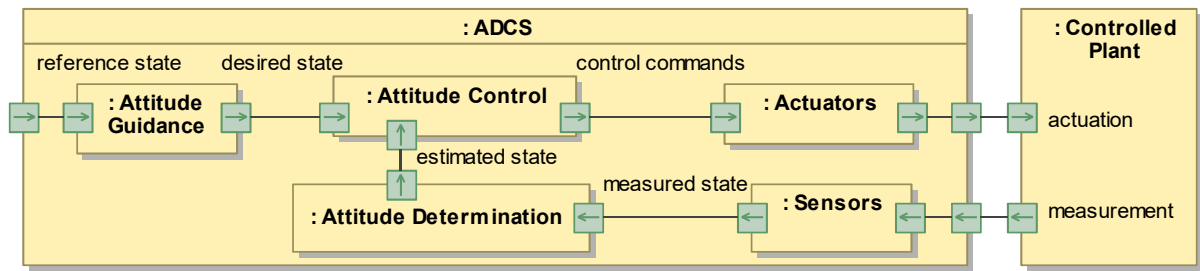


Figure 2.9: Structure of an ADCS based on active control techniques.

normal mission, used for regular mission operation; (ii) slew, i.e., reorienting the vehicle; (iii) safe mode, used in emergencies; and (iv) detumbling, i.e., to null all angular velocity components of the spacecraft. Depending on the mission objectives, the ADCS may be designed to stabilize a spacecraft relative to an inertial frame or relative to a rotating frame. The former applies to spacecrafts designed to observe distant stars, for example, while the latter applies to spacecrafts designed for Earth observation.

### 2.6.1 Sensors

The sensors used for the determination of the spacecraft attitude either measure the angular velocity of the spacecraft or a known vector quantity, i.e., a vector quantity that can be estimated in the inertial frame. These two types of measurements can then be integrated to determine attitude with high accuracy at a high enough frequency. The measurement of a known vector quantity is always coupled with the estimation, in the inertial frame, of that same quantity. There are several types of attitude sensors:

**Gyroscope** Gyroscopes measure either the speed or angle of rotation of the spacecraft relative to an inertial reference. There are three types of gyroscopes: spinning mass gyros, optical gyros, and Coriolis vibrating gyros (CVGs) [71]. Spinning-mass gyros rely on the conservation of angular momentum of a rotating mass relative to an inertial reference, optical gyros are based on the *Sagnac* effect, and CVGs rely on vibrational modes being induced in structures by the Coriolis effect. CVGs based on MEMS technology are commonly used in CubeSats because they are small and have a low cost, mass, and power consumption. However, they suffer from the problem of drift due to thermal variations [72], which may be compensated with attitude filtering.

**Magnetometer** Magnetometers are widely used as spacecraft attitude sensors [73]. These are specifically used as attitude sensors in LEO orbits, since a well-modelled magnetic field is required [71]. They are reliable and have a low mass and power consumption. The measurements of the magnetometer are easily disturbed by the magnetic dipole induced by magnetic torquers. Thus, magnetometer measurements must alternate with magnetic torquer actuation intervals. The simplicity and low cost of this type of sensor contrast with a lower accuracy, when compared with star or horizon sensors, for example. The accuracy of a magnetometer ranges from  $0.5^\circ$  to  $3^\circ$  [22] but, since the Earth magnetic field vector cannot be precisely computed, the actual accuracy is lower. For the computation of the Earth magnetic

field vector, a precise estimate or measurement of the spacecraft position is required.

**Sun Sensor** Sun sensors can be divided into two categories: coarse sun sensors (CSSs) and fine or digital sun sensors (DSSs) [71]. CSSs are usually photodiodes, which output an electric current directly proportional to the cosine of the Sun angle. For an accurate Sun vector estimate to be derived, at least three CSSs must have the Sun in their field-of-view (FOV). DSSs are larger, more expensive, and heavier, as well as more accurate and reliable. The Sun vector can be estimated using only one DSS if the Sun is inside its FOV. The determination of the Sun vector in the inertial frame may be accomplished by on-board propagation of the Earth's orbit around the Sun. LEO orbits usually include eclipse periods, and the regular loss of the Sun vector data may compromise the ability to meet pointing requirements. CSSs have a measurement error of up to  $3^\circ$  [22], but the influence of the Earth albedo can severely degrade their accuracy. In contrast, DSSs can be quite accurate, with measurement errors being as low as  $0.01^\circ$  [22].

**Earth Sensor** Earth sensors (also called horizon sensors) are infrared devices which can detect the “contrast between the cold of deep space and the heat of the Earth's atmosphere” [22]. These detect points on the Earth's horizon which are then used to derive a nadir vector. Earth sensors require an unobstructed FOV and, for complete coverage, two or more sensors are usually required, since the FOV of the sensor is typically 180 degrees or less. The nadir vector can be determined in the ECI frame directly from the estimate of the spacecraft position. Earth sensors are specifically appropriate for Earth-pointing spacecrafts, for which only one sensor may be enough if mounted appropriately. The accuracy of this type of sensor can range from  $0.1^\circ$  to  $0.25^\circ$  [22].

**Star Tracker** Star trackers track one or more stars to derive attitude information. These are basically digital cameras which allow the light from stars to fall on a CCD (charge-coupled device) or an APS (active pixel sensor), creating an image of the star field [22]. These sensors are very accurate, with errors ranging from 0.0003 to 0.01 degrees [22], but they are also usually heavier, larger, and more expensive than other types of attitude sensors.

## 2.6.2 Attitude Determination

Determining the attitude of a body requires finding three independent scalar quantities, such as any minimal parametrization of the attitude matrix [71, p. 183] (e.g., the three Euler angles). The first ever published attitude determination methods were static [71] (e.g., TRIAD algorithm), i.e., they relied on two or more vector measurements to algebraically compute an attitude estimate without making explicit use of knowledge of the spacecraft motion.

Since then, using attitude filters has become the most common approach, with these providing more accurate attitude estimates than static methods [71]. Vector measurements are combined with a rotational kinematics model, which is propagated using the angular rates that are measured with a gyroscope. As mentioned before, gyroscope measurements suffer from drift, which causes the attitude

propagation to quickly diverge. The vector measurements are used by the attitude filters to correct the attitude propagation and estimate the gyroscope drift.

Early applications mostly used extended Kalman filters for determining attitude [74]. Several schemes using Kalman filters exist, differing on their treatment of the attitude error, some of which are examined in [75]. Since then, new approaches have been developed, some of which have proven to be superior to the extended Kalman filter (e.g., filter QUEST, extended QUEST, backwards-smoothing extended Kalman filter) [74]. The choice of an appropriate attitude determination method depends on many variables, including the required accuracy, vehicle properties, sensors, and available resources (e.g., processing power or memory).

### 2.6.3 Actuators

Attitude actuators can be divided into two categories: momentum-exchange devices, which conserve angular momentum (e.g., reaction wheels), and external torque actuators, which change the angular momentum of the spacecraft by producing external torques (e.g., magnetic torquers or thrusters) [22].

**Magnetic Torquer** Magnetic torquers consist of magnetic coils that create a magnetic dipole which interacts with the Earth magnetic field, resulting in an applied torque that is orthogonal to both the magnetic dipole and the magnetic field vector [73]. These can be used for attitude control but also to unload angular momentum accumulated by reaction wheels. Magnetic torquers are simple and inexpensive and they are especially used in LEO, where the Earth magnetic field intensity is higher. Three torquers mounted orthogonally can create a magnetic dipole with any direction, but at any given moment, no torque can be provided about the Earth magnetic field vector. As the latitude or altitude vary, the magnetic field vector changes its direction, providing new directions for the actuation torque [22]. For the operation of a magnetic torquer, the Earth magnetic field vector must be known in the body frame, for example, using magnetometer readings.

**Reaction Wheel** Reaction wheels are the primary attitude control actuators of most spacecrafts [71]. These are basically torque motors with high-inertia rotors [22]. Each reaction wheel provides one axis of control, which means three wheels with non-coplanar axes are required for 3-axis control. Sometimes one extra wheel is also added for redundancy. The two most relevant performance attributes of reaction wheels are the angular momentum capacity and the maximum actuation torque (which varies with the wheel angular rate). The angular momentum capacity is usually sized to enable the reaction wheels to counteract the full cyclic disturbance components without requiring frequent momentum dumping [22]. The momentum accumulated due to secular components of the disturbances can in some cases be dumped from time to time, using an external torque actuator. The maximum actuation torque is usually constrained by slew requirements of the ADCS [22]. Reaction wheels provide a more precise attitude control, when compared to magnetic torquers, but they are heavier, larger, and more expensive [76].

**Control Moment Gyros** Control Moment Gyros (CMG) are “single- or double-gimbaled wheels spinning at constant speed” [22, p. 580]. High output torques can be obtained by turning the gimbal axis and two CMGs can produce torques about all three axes of the body frame. This type of actuator is used for high-torque and fine control applications only, because they: (i) are expensive and heavy; (ii) have short lifetimes and high-power needs; and (iii) require complex control laws and careful momentum management to avoid wheel saturation [22].

**Thruster** Thrusters produce a force by “expelling material, called propellant, at high velocity from their exit nozzles” [22, p. 580]. These can be used for both trajectory control and attitude control, which makes them a viable option for spacecrafts that both require orbital manoeuvres or corrections and attitude stabilization. The torque provided by a thruster is proportional to their moment arm, which means that it is both limited by the available force and the physical dimensions of the spacecraft. The disadvantages of this type of actuators are the use of expendable propellant and the difficulty of obtaining, for a given thruster configuration, both high-torque and fine attitude control.

#### 2.6.4 Attitude Guidance and Control

For each mode of an ADCS, different objectives are defined, and distinct guidance and control algorithms may be required. The ADCS design of a nanosatellite usually includes detumbling algorithms that are simple and extremely reliable. This way, essential functions of the spacecraft that cannot be performed under high angular rates are not exposed to failure of the more complex ADCS algorithms or hardware.

Control algorithms for 2-axis or 3-axis attitude stabilization are often more complex. Attitude guidance is designed according to the specific objectives of the mission, usually translating some reference state into a desired attitude and angular velocity. These last two are inputs to the attitude control algorithms, whose design depends on the characteristics of the spacecraft, the mission objectives for each mode of operation, and the selected actuators.

Algorithms based on proportional-derivative (PD) control are among the simplest algorithms that can be used for attitude control. This type of algorithm has for a long time been shown to be effective for attitude control based on unrestricted actuation [77]. In the case of using magnetic torquers, which are restricted in terms of actuation direction as mentioned above, the accuracy of this type of control algorithm may be insufficient. Nonetheless, ADCS designs comprising magnetic torquer actuation and PD attitude control are proposed in literature [78–80]. Multiple methods are used for the definition of the PD gains, such as linear-quadratic regulation (LQR) [79]. For cases in which stability margin requirements are easy to meet, but accuracy requirements are difficult, a proportional-integral-derivative (PID) controller may be used instead, to remove the steady-state attitude error [22].

More complex attitude control algorithms have also been proposed, e.g., based on model predictive control techniques or sliding-mode control techniques. In addition to the accuracy of the attitude control, some other performance indicators are also commonly studied, such as stability margins (e.g., gain margin and phase margin) or the level of robustness (i.e., “the size of the uncertainty domain over which control objectives are achieved” [81]).

# Chapter 3

## Approach

### 3.1 Methodology Adaptation

In this work, we propose an MBSE methodology that is tailored for the design of a spacecraft subsystem, and we apply it to a design process following the definition in Section 2.5. This methodology results from the adaptation and extension of the OOSEM (discussed in Section 2.3). The modifications made to the OOSEM aim to adapt it to the design of a subsystem while reducing the overall complexity of the design process. Also, the OOSEM is extended to include the integration of simulation into the design process, which is implemented with a model transformation approach that leverages the SysPhS standard. This approach results in the generation of a complete Functional Engineering Simulator. To this end, a framework capable of generating simulation models from SysML specification is developed.

The integration between MBSE and simulation constitutes an interaction between systems engineering and other specialized domains. Systems engineers cannot develop a SysML model that caters to the needs of the domain specialists and the latter cannot migrate their domain-specific models to SysML. Instead, the SysML system model can be integrated with the domain-specific models.

With this type of approach, we consider that a set of “subsystem engineers” are required, whom must have a good understanding of SE practices and master the use of MBSE tools and languages, while having a proper knowledge of one specialty field. These engineers are responsible for specifying the design of one subsystem in SysML and implementing the integration between SysML and the simulation environment. Teams of specialist engineers of the same field may then develop and maintain mathematical models and execute simulations using a dedicated simulation environment, as well as perform other activities such as validating the models or developing and testing hardware and software components.

Since the focus of this work is the design of a subsystem, its scope does not include the application of the *Analyse Stakeholder Needs* activity and all the other OOSEM activities at the system-level. However, since we aim to design a subsystem in the context of a complete system design process, the artifacts that result from the application of these activities are a pre-condition to this work (e.g., the decomposition of the system into subsystems provides necessary context for the design of each subsystem).

To ensure a reduction in the modelling effort required to generate these artifacts, the model developed in [21] was used as a basis for the model developed in this work. The model in [21] results from



the application of OOSEM to the development of a preferred system architecture of a spacecraft. The reuse of this model reduces the total modelling effort because the organization of the model and some of its elements can be reused, including the mission- and system-level specification, as they are equally valid in the context of this work. Modifications were made to the model organization, due to methodology changes, and to the mission- and system-level specification, reflecting the differences between the design scenario implemented in this work and the one implemented in [21].

### 3.1.1 Process Adaptation

The OOSEM process, shown in Figure 2.4, was modified to become less complex and better suited for the design of a subsystem, resulting in the subsystem definition process represented in Figure 3.1. The modifications applied to the OOSEM process are described in this section.

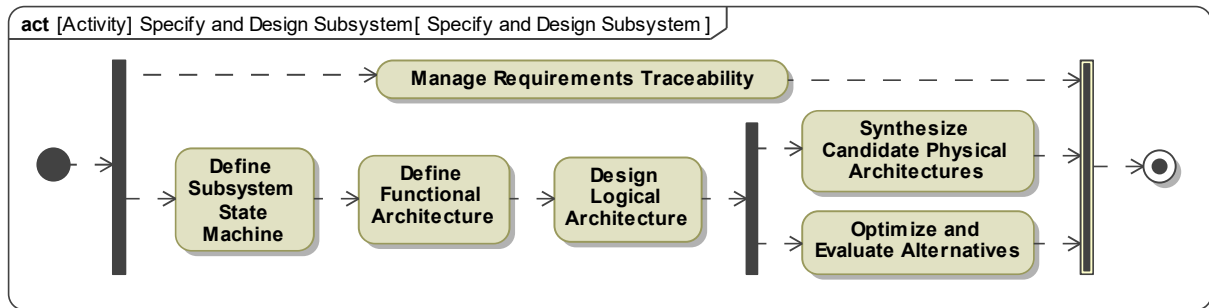


Figure 3.1: Subsystem design methodology process (each node represents an activity or task).

First, in a recursive application of the OOSEM process, the *Synthesize Candidate Physical Architectures* activity applied at the system-level contains the same tasks as the *Analyse System Requirements* activity applied at the subsystem-level (see Figure 2.4), except for the specification of the subsystem state machine behaviour. To resolve this redundancy, we concentrate at the system-level all the tasks in common, so that their application becomes a pre-condition to the subsystem design process. The reason for this is that some of the inputs to the design process defined in Figure 2.8 result from the application of these activities. Thus, in the final methodology process, the *Analyse System Requirements* activity is replaced by the task of specifying the subsystem state machine behaviour (see Figure 3.1).

Additionally, the OOSEM defines two levels of component abstraction, but we consider that three levels are required for the definition of the subsystem-level of the architecture (an approach also followed in [47]). The OOSEM defines a logical level, in which components are defined in terms of their functionality, without the imposition of implementation constraints; and a physical level, in which components are defined as specific implementations of the available technology.

To support the view that three levels are required, we may consider a specific example. In an early design phase, the need may be identified for including a momentum accumulator device in the ADCS design (first level of abstraction). Later, the available technology at a specific moment in time is considered and different types of momentum accumulator devices are compared, resulting in the selection of one of them, e.g., reaction wheels (second level of abstraction). Many different types, sizes, and

manufacturers of reaction wheels are available, so one final level of abstraction is required to represent the selection of either a particular procured model or a specific in-house design of a reaction wheel.

In this work, we define the three layers as the *functional*, *logical*, and *physical* architectures of the subsystem. The definitions used are: (i) the functional level consists of the definition of function (equivalent to the logical level defined in [14]); (ii) the logical level consists of a particular implementation of the available technology (equivalent to the physical level defined in [14]); and (iii) the physical level consists of the selection of a particular procured or developed component. Since the layer definitions presented in this work and in [14] are not consistent, hereinafter the definitions used are those presented here.

One final modification to the OOSEM was made, addressing the fact that performing analysis at all the levels of abstraction is too complex for the scope of this work. The architecture layers are usually defined sequentially and recursively, and for all these, candidate architectures can be synthesized and compared using some type of analysis. To reduce the complexity of the design process, architecture alternatives are only analysed at the physical level of abstraction, where the characteristics of the components can be easily determined (e.g., through the specification of COTS solutions). For more abstract layers, these characteristics can be determined by using pre-existing designs as a reference. For example, the precision of a magnetic torquer-based attitude controller may be estimated by evaluating known ADCS designs that use this type of actuator, before the actual precision is determined with simulation.

### 3.1.2 Activity Definition

In Figure 3.1, the *Define Subsystem State Machine* node represents a task, and the remaining nodes represent activities (which are composed of multiple tasks). This task is applied in this methodology as defined in [14] and the remaining activities are defined in this section.

In this methodology, the *Manage Requirements Traceability* activity is defined exactly as in [14] (see Figure 3.2). The *specification tree* is defined first, providing an organizational scheme for the definition of text-based requirements at each level of the architecture. As requirements are captured in the model, relationships are established between these and other requirements in the *specification tree* (namely *derive requirement* and *refine* relationships), as well as with other types of elements (namely *satisfy* and *verify* relationships). Examples of these relationships are shown in Figure 2.2. During the complete design process, as requirements are updated and traceability gaps are identified, iterations will occur to redefine the requirements or the relationships defined previously.

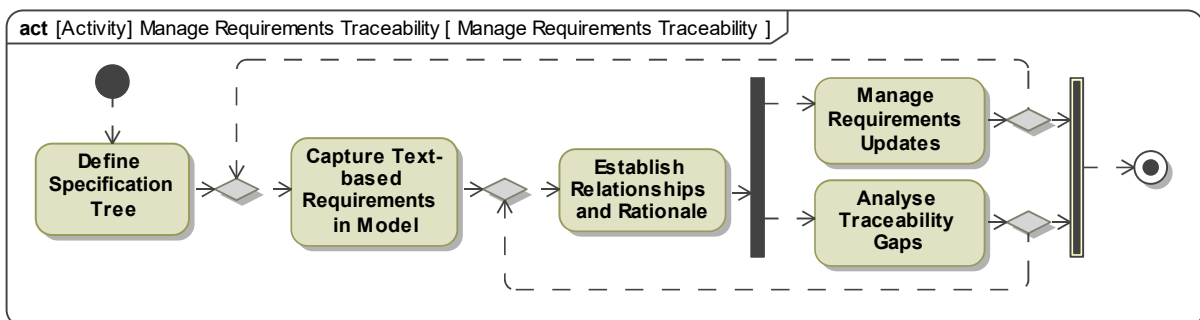


Figure 3.2: *Manage Requirements Traceability* process (each node represents a task).

With the scope defined for this work, we are able to demonstrate the definition of the *specification tree* and the capture of text-based subsystem requirements. The relationships between the requirements and the elements that satisfy and verify them are also modelled. Traceability between subsystem requirements and higher-level requirements is demonstrated for a subset of the subsystem requirements.

The definition of the functional architecture of the subsystem, represented in Figure 3.3, is based on the process defined in [14] for the *Define Logical Architecture* activity. The subsystem is first decomposed into functions (or functional components), after which a set of interactions between these are modelled to realize the previously specified behaviours of the subsystem (which are usually modelled with *activities*). The interactions that are defined clarify how the functional components should be connected to each other, facilitating the task that follows, which consists of specifying the internal structure of the subsystem using an **ibd**. The final step is the specification of each functional component as a black box, part of which results directly from the application of the previous two tasks.

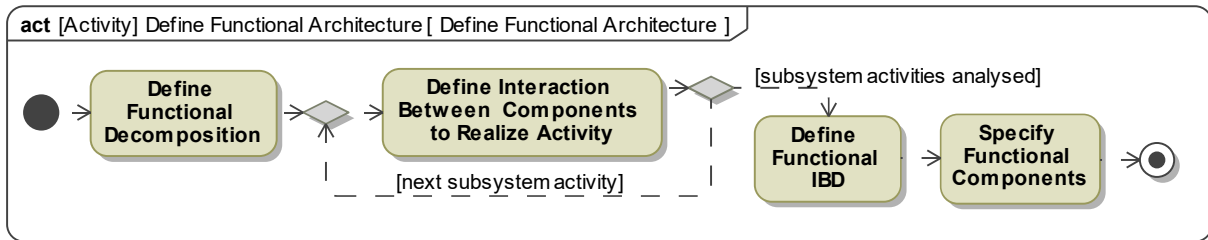


Figure 3.3: *Define Functional Architecture* process (each node represents a task).

The design of the logical architecture follows a process as shown in Figure 3.4, which is based on the one defined in [14] for the *Synthesize Candidate Physical Architectures* activity. This activity starts with the definition of the logical architecture of the subsystem, which follows the process shown in Figure 3.3, but at the logical level of abstraction. After the definition of the logical architecture, software and hardware architectures of each subsystem are defined and integrated with those of the other subsystems. This results in software and hardware architectures defined at the subsystem level but extending across the complete system. Their specification ensures that the design of each subsystem is well integrated with the other subsystems. The final step in this process is the specification of the requirements for each logical component, which results from the application of the tasks already mentioned, but also from the continuous capture of critical component properties during the complete application of this activity.

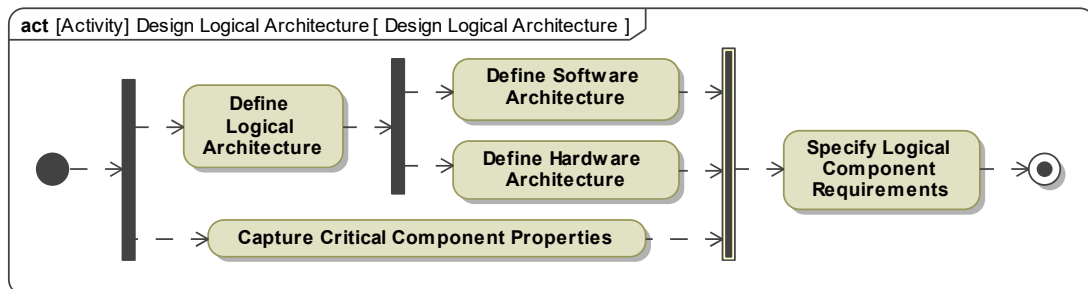


Figure 3.4: *Define Logical Architecture* process (each node represents a task).

The process used for synthesizing candidate physical architectures is represented in Figure 3.5.

According to the needs of the project, a set of candidate physical architecture are defined. These are then analysed and compared, and a preferred architecture is selected. The physical architectures do not differ from the logical architecture in terms of structure or behaviour. The definition of the candidate physical architectures consists solely of attributing values to the *value properties* of the components of the subsystem and redefining element multiplicities. In this work, the different candidates will only differ in terms of element multiplicities, i.e., different options of components are not compared. This is because comparing architectures that differ in their composition is more interesting to the validation of the methodology than comparing architectures with different component options. In fact, to compare different component options, we only change the values of some of the attributes and repeat the exact same model transformation process (used to generate a simulation model).

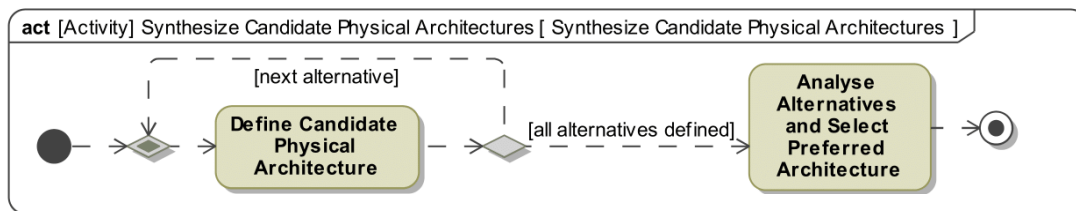


Figure 3.5: *Synthesize Candidate Physical Architectures* process (each node represents a task).

The *Optimize and Evaluate Alternatives* activity requires the support of simulation which, as mentioned in Section 2.4, must be performed inside the scope of a dedicated simulation environment. The process for this activity, which is shown in Figure 3.6, is based on the one defined in [14]. All the required analyses are identified, and for each of these an *analysis context* is defined. An *analysis context* is a block that is composed of all the elements relevant to the analysis, including the complete SysML specification of the simulation model's internal structure and of the behaviour of all the elements contained in its architecture. While the process for this activity is similar to the one defined in the OOSEM, the methods used to implement the tasks of defining the *analysis context* and performing engineering analysis must be extended to include the integration between the MBSE modelling tool and a simulation environment. These methods are discussed in Section 3.1.3.

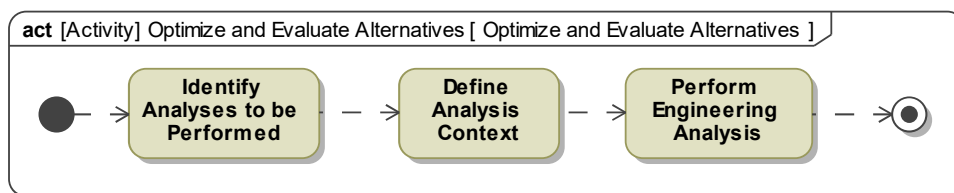


Figure 3.6: *Optimize and Evaluate Alternatives* process (each node represents a task).

### 3.1.3 Integrating MBSE with Simulation

System simulation has many potential uses in support of engineering activities across the system life cycle, some of which are presented in [60]. In this work, simulation is essential to support the early verification of requirements, especially performance requirements.

The methodology proposed in this chapter addresses the design of a spacecraft subsystem, and most of its activities focus on the definition of the subsystem architecture and on the specification of subsystem design configurations. Both of these are developed according to requirements defined for the subsystem. In this context, analysis must be used to compare the performance of different designs or to verify if a specific design is valid (i.e., satisfies the requirements), which in the case of complex dynamic behaviour usually is performed with simulation. In this methodology, all analysis activities are integrated into the *Optimize and Evaluate Alternatives* process.

In the context of MBSE, the capability of performing simulation consists of being able to execute the specification contained in the system model. The integration of the MBSE tool with a simulation environment is intended to implement this execution of the system specification. In this work, this integration is implemented with a model transformation approach, which aims to transform SysML specification into a different type of specification based on executable semantics, essentially generating a simulation model that can be executed. For the part of the specification that cannot be transformed, due to semantic incompatibility, it must be replicated manually using executable semantics, and the latter may then replace the former in the simulation model.

As mentioned in Section 2.4, the objective of this integration is to generate a Functional Engineering Simulator that can be executed to generate values for TPMs. In this section we propose a method for this integration, which applies specifically to the case of generating an individual simulation model from a single *analysis context* specified in SysML. The *analysis context* can be exported to the simulation environment, in this case MATLAB/Simulink, as a simulation-ready model.

### Overall Model Transformation Approach

Before the transformation is performed, the system, the environment, and the *analysis context* must be specified in SysML, and the mathematical models must be created in Simulink. To do this, we must:

1. Model the components of the system and environment in SysML and Simulink.
2. Specify the system and environment in SysML using **bdd**, **ibd**, **stm**, and **par** diagrams.
3. Specify the *Monitoring and Control* block (see Figure 2.5).
4. Specify the *analysis context*, which integrates the system with the environment and the *Monitoring and Control* block.
5. Define values for the attributes of the system and environment, and for the general simulation parameters.
6. Create tables outlining key aspects of the SysML specification and synchronise these with Excel spreadsheets (more details are provided later in this section).

The process of performing engineering analysis consists of applying the model transformation process and executing the simulation model. The model transformation process uses the *Simulink Export*

function of CSM, which implements the SysPhS standard, and a MATLAB program that was developed to extend this CSM function. In the point of view of the user, the following actions are taken:

1. Ensure that the Excel spreadsheet synchronization is up to date.
2. Select the *analysis context* in the CSM tool and press the *Simulink Export* button.
3. After the Simulink model is generated by CSM, execute the MATLAB program.
4. Run the Simulink model (the TPM values will be automatically generated and saved).

The model transformation workflow is represented in Figure 3.7. The *Simulink Export* function applies to any block in the model, exporting a simulation model which replicates its internal structure. The internal structure in this case comprises the decomposition of the block into other blocks, the external interface of the block and its composing elements (i.e., their *ports*), and the connection between any two *ports* owned by composing blocks. The *analysis context* is exported using this function (identified as step 1 in Figure 3.7). All the components contained in the *analysis context* are exported as black boxes, or empty Simulink blocks, since this is how they are defined in SysML.

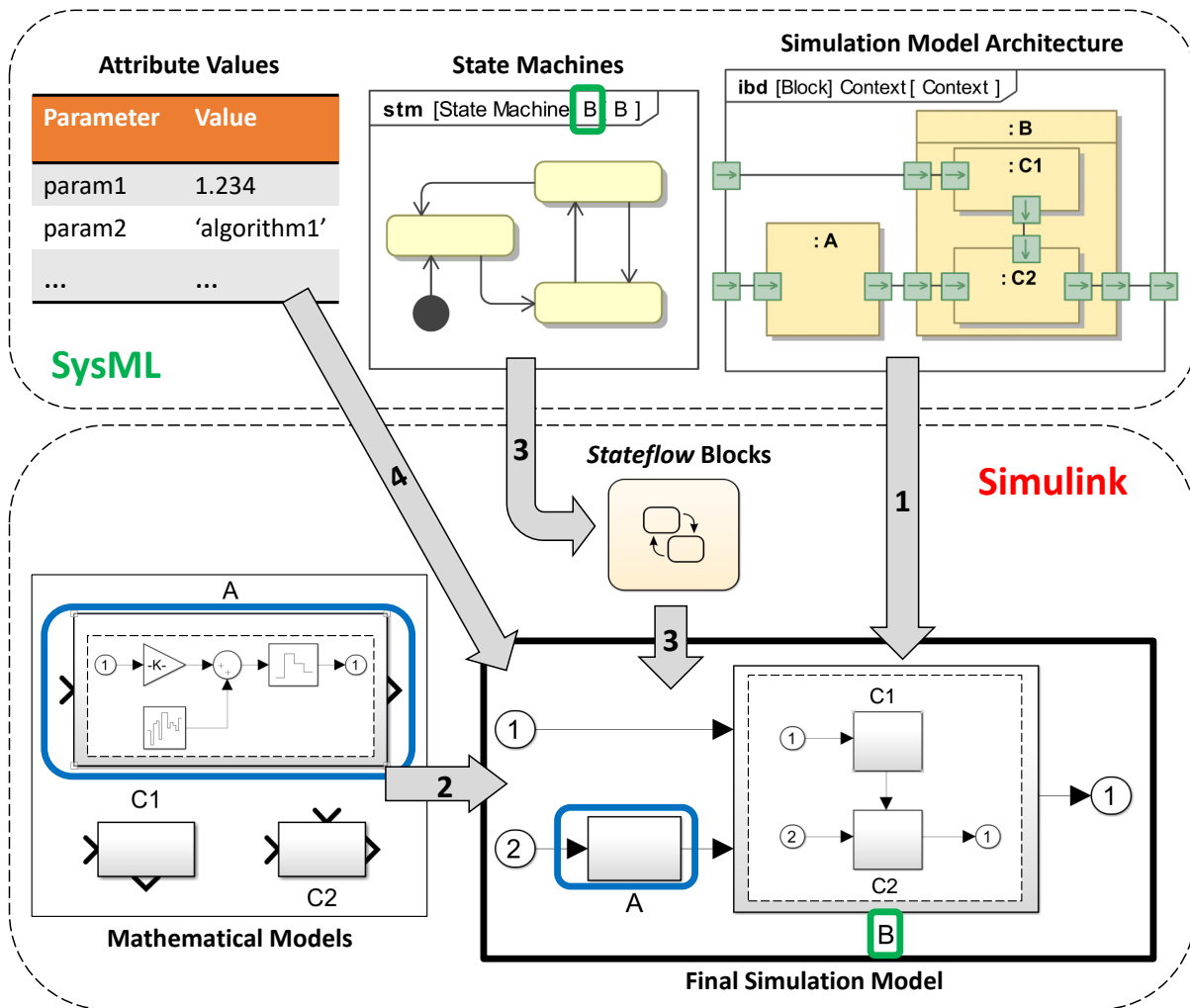


Figure 3.7: Model transformation workflow. The large grey arrows represent steps in the transformation.

The CSM function can generate the “skeleton” of the simulation model, but the rest of the transformation must be performed by the developed MATLAB program. Mathematical models should be developed inside the scope of the simulation environment, but these can only be created to represent the behaviour of the lower-level elements, i.e., the components. These mathematical models, which are created in Simulink as *Subsystem* blocks, replace the exported empty Simulink blocks in the process of generating the final simulation model (step 2 in Figure 3.7). Libraries are created both in SysML and Simulink, containing representations of hardware and software components, which may either be COTS or developed in-house. These libraries can and should be reused across different projects, contributing to the reduction of project resource expenditure.

With this model transformation approach, most of the behaviour of the system and all its composing elements, in specific that which is relevant for simulation, is first decomposed down to the component-level. At this level, the behaviour of the elements can be specified in SysML, while a mathematical representation of the behaviour is also created in Simulink to be used in simulation. However, the behaviour of complex systems in general cannot be completely decomposed into behaviour of the components. To address this issue, this approach also enables the transformation of SysML state machines into Stateflow blocks defined in Simulink (step 3 in Figure 3.7).

SysML state machines specify the behaviour of elements in terms of the states in which they may be. Multiple states can be specified as well as the behaviour of the element in each of its states. Transitions between states are also specified in terms of the events that trigger them, the conditions that must be verified for them to occur, and the effect they have regarding the behaviour of the element. This type of behaviour specification is essential in a simulation model, since the behaviour of the elements involved in the simulation may depend on the state in which other elements are in each instant.

Parameter values are transferred from SysML specification into the simulation model, ensuring that all the quantitative attributes of the elements can be both specified in SysML and accessed directly by the simulation environment (step 4 in Figure 3.7). In the rest of this section, a closer look is taken into each specific aspect of the model transformation approach.

## Modelling Internal Structure

The internal structure of the *analysis context* and its composing elements, specified with **ibd** diagrams, can be easily replicated into a Simulink model using the CSM *Simulink Export* function. However, this standard introduces some constraints into the SysML modelling practices. First, it requires that *ports* have only one direction and that they be typed by *interface blocks*. These are composed of *flow properties*, which define the types of entities that flow through the *port*. Additionally, two blocks may not interact with each other via a connection between *ports*, without this connection passing through all the intermediate levels of the architecture (an example of this is available in Figure 4.3, concerning the connections between the ADCS and the *SC Rotation* block).

## Modelling Components

Components are modelled in SysML as black boxes, and so they only contain a set of *ports*, *value properties*, and allocated behaviours. Since these are modelled both in SysML and Simulink, a mapping between these two representations is required. As a default, a block in SysML is mapped to a *Subsystem* block in Simulink that has the same name, if one exists. However, this default mapping can sometimes be restrictive, and so the property *owned comment* of a block can be used to define the name of a different *Subsystem* block that models its behaviour, overriding the default mapping.

The *ports* owned by the block must match the inputs and outputs of the *Subsystem* block both in their order and directions. Also, all the time-independent parameters that are used in the *Subsystem* block, must be defined in SysML as *value properties* of the component block (see Figure 3.8). When changes must be made to a component, these changes are propagated automatically through the model, but consistency between the SysML and Simulink black box representations must be ensured manually.

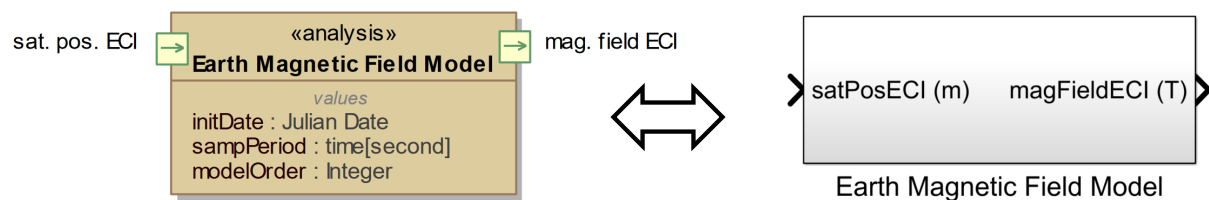


Figure 3.8: Example of the mapping between SysML and Simulink representations of a component. The environment is also divided into components, which are identified with the *stereotype* «analysis».

## Defining the scope of the simulation model

The MATLAB program connects the raw simulation model exported by the CSM with the mathematical models stored in a Simulink library. For this, the user must identify all the elements whose Simulink representation is relevant to the simulation model in question. To this end, a table is generated outlining each *part property* defined in the model in terms of: (i) its name, used to uniquely identify any element in the architecture; (ii) the name of the block that types it, which is necessary to know what mathematical model represents its behaviour; and (iii) the name of the *instance specification* that is defined as its default value, which specifies values for the attributes of the specific element (see Figure 3.9).

| # | Name      | Type                       | Default Value  |
|---|-----------|----------------------------|----------------|
| 5 | aeroTMod  | Aerodynamic Torque Model   | aeroTMod_spec  |
| 6 | albMod    | Earth Albedo Model         | albMod_spec    |
| 7 | magFldMod | Earth Magnetic Field Model | magFldMod_spec |

Figure 3.9: Segment of the table outlining the *part properties* relevant to the simulation model.

## Defining Parameter Values

With the same simulation model, different simulations for different system configurations may be performed by changing the values of the internal parameters. These internal parameters are specified as



*value properties* of the component blocks, as mentioned above, and they must be completely defined inside the SysML model, to ensure the centralization of information into a single source of truth.

Some *value properties* in the model are related to each other, representing the same quantitative attribute. For example, the *Spacecraft Rotation Dynamics* block owns a *value property* that represents the moment of inertia of the spacecraft, a parameter necessary to compute its attitude and angular velocity. However, this parameter is also represented by a *value property* of the *Spacecraft* block, since this is a quantitative attribute of the spacecraft as a whole. In these situations, **par** diagrams are used to define *binding connectors* between the two properties, which define a mathematical equality relationship.

The MATLAB program retrieves all the relevant information for the definition of parameter values and introduces these into the exported Simulink model. To support this process, a table as the one in Figure 3.10 is created to outline all the values defined with *instance specifications*. The information in this table relates each *instance specification* to the *value properties* defined by it, including the values attributed to each property. When *value properties* are connected in **par** diagrams, a value is attributed to one of these, while the other is presented in the table without a specified value. By processing the specification of the *binding connectors* mentioned above, the MATLAB program fills in the gaps automatically.






| #  | Owner                                                                                                                        | Defining Feature                                                                                              | Value                                                                                           |
|----|------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| 44 |  albMod_spec : Earth Albedo Model           |  initDate : Julian Date      |                                                                                                 |
| 45 |  albMod_spec : Earth Albedo Model         |  sampPeriod : time[second] |  -2.0 10.0 |
| 46 |  aeroTMod_spec : Aerodynamic Torque Model |  CD : Real                 |                                                                                                 |
| 47 |  aeroTMod_spec : Aerodynamic Torque Model |  sampPeriod : time[second] |  -2.0 1.0  |

Figure 3.10: Segment of the table outlining the values defined with *instance specifications*.

### Transformation of SysML State Machines into Stateflow blocks

The semantics of SysML state machines are very extensive [14], so the transformation of state machines into Stateflow blocks that is developed here supports only the most relevant parts of these semantics. The mapping used between SysML and Stateflow constructs is presented in Table 3.1, and it is based on the mapping defined by the SysPhS. The MATLAB program that implements this transformation uses the Stateflow API [82] to create and specify the Stateflow blocks inside the Simulink model.

| SysML                                      | Stateflow          |
|--------------------------------------------|--------------------|
| State                                      | State              |
| Transition                                 | Transition         |
| <i>Junction Pseudostate</i>                | Junction           |
| Transition from <i>initial pseudostate</i> | Default Transition |
| <i>Guard</i>                               | Condition          |
| <i>Relative TimeEvent</i>                  | after() statement  |
| <i>SignalEvent</i>                         | Message            |

Table 3.1: Mapping between SysML state machine and Stateflow constructs.

Some SysML constructs are not supported by this model transformation approach, such as multiple

regions inside a state machine, composite states, *absolute time events*, and *call events*. An alternative to modelling composite states that is compatible with this model transformation method is to specify the behaviour of a state with its own state machine.

The process of transforming state machines into Stateflow blocks starts with the identification of all the state machines in the model that may be relevant for the simulation. These are listed in a table, such as the one shown in Figure 3.11 (left), where the name of the state machine is matched to an *owner block* and to an *owned comment*. The MATLAB program adds a Stateflow block to every Simulink block representing an element whose behaviour is specified by one of the state machines that are listed.

In SysML, a block owns the state machine that represents its classifier behaviour as well as all the state machines that specify states of this behaviour. To create a distinction, the *owned comment* property is used to specify if the state machine represents a classifier behaviour or the behaviour of a state. The string “CB” indicates that the state machine represents a classifier behaviour (see Figure 3.11). Each state machine contains only one active state at a time, which is held in a variable that is matched to each state machine and outlined in a table, as shown in Figure 3.11 (right).















| # | △ Name                                                                                                       | Owned Comment                                                                          | Owner                                                                                          | # | Name                                                                                                         | Owned Comment                                                                                 |
|---|--------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|---|--------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| 1 |  ADCS Main Behavior         |  CB   |  ADCS         | 1 |  ADCS Main Behavior         |  adcsMode  |
| 2 |  OnState                   |                                                                                        |  Spacecraft  | 2 |  OnState                   |  scMode   |
| 3 |  Spacecraft State Machine |  CB |  Spacecraft | 3 |  Spacecraft State Machine |  scState |

Figure 3.11: List of state machines relevant to the simulation model (left) and correspondence between each state machine and the name of the variable that stores its active state (right).

For each Stateflow block added to the simulation model, the MATLAB program first creates all the states contained in the state machine. All the states contained in the relevant state machines are outlined in a table, as shown in Figure 3.12. The *entry*, *do*, and *exit* behaviours specified in SysML, and outlined in this table, are attributed to the state in Stateflow if they are typed by an *opaque behaviour* (a behaviour expressed as a set of language-specific statements). An additional entry action is added to each state in a Stateflow block defining the value of the variable which holds the active state information.











| # | Name                                                                                          | Owner                                                                                                        | Owned Comment                                                                         | Entry | Behavior Type | Do Activity | Behavior Type | Exit | Behavior Type | Submachine                                                                                    |
|---|-----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|-------|---------------|-------------|---------------|------|---------------|-----------------------------------------------------------------------------------------------|
| 1 |  off       |  Spacecraft State Mach... |  0 |       |               |             |               |      |               |                                                                                               |
| 2 |  on        |  Spacecraft State Mach... |  1 |       |               |             |               |      |               |  OnState |
| 3 |  launching |  Spacecraft State Mach... |  2 |       |               |             |               |      |               |                                                                                               |

Figure 3.12: Segment of the table outlining the states contained in the relevant state machines.

The integer value attributed to each state is outlined in the table shown in Figure 3.12. As an example, in the Stateflow block representing the behaviour of the spacecraft, when the *on* state is entered, the value of the variable *scMode* is changed to 1. Also, the active state of a Stateflow block must be accessible externally. To ensure this, an output *data object* is created for the variable that specifies the active state. This output is connected to a *goto flag* with the name of the variable, making it globally accessible in the simulation model. Figure 3.13 shows a sample Stateflow block that is used here to

exemplify some of the relevant aspects of this transformation. In this example, the variable that identifies the active state of the state machine *STM1* is called *activeState\_STM1*. As mentioned above, an output of the Stateflow block is created for this variable, and it is connected to a *goto* flag.

The table in Figure 3.12 also identifies the states that are specified with a *submachine*, so that the MATLAB program can identify the states that are specified with a state machine. This program is capable of specifying the internal elements of this state exactly as if it were a new Stateflow block.

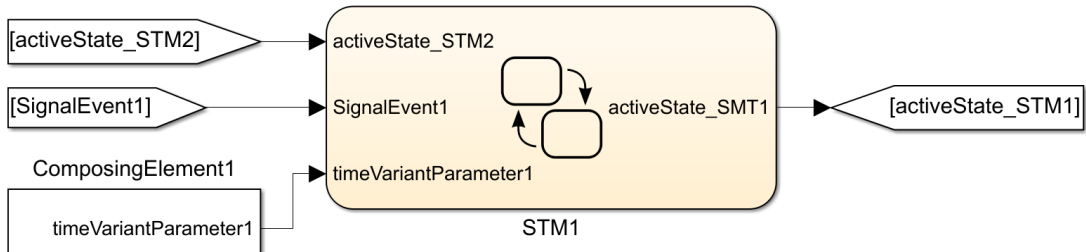


Figure 3.13: Example of a Stateflow block generated from SysML specification.

A different table is used to summarize all the pseudostates used in each state machine. *Initial pseudostates* are defined in Stateflow as *default transitions*, *junction pseudostates* are replaced by *junctions*, and *forks* and *joins* are suppressed when the Stateflow block is generated. This is done by redefining either the source of the incoming transitions or target of the outgoing transitions, respectively. After the states and pseudostates have been created inside the Stateflow block, transitions can be created according to the transition specification outlined in a table as the one shown in Figure 3.14.

| Owner                    | Event Element                                              | Guard | Effect | Behavior Type | Source     | Target     |
|--------------------------|------------------------------------------------------------|-------|--------|---------------|------------|------------|
| Spacecraft State Mach... | <input checked="" type="checkbox"/> SignalEvent Power_Off  |       |        |               | on:OnState | off        |
| Spacecraft State Mach... | <input checked="" type="checkbox"/> SignalEvent Power_On   |       |        |               | off        | on:OnState |
| Spacecraft State Mach... | <input checked="" type="checkbox"/> SignalEvent Reboot_Cmd |       |        |               | on:OnState | rebooting  |

Figure 3.14: Segment of the table outlining the transitions contained in the relevant state machines.

The transitions are added to the Stateflow block (or state), matching the source and target states or pseudostates identified in the table. The *guard* of a transition is defined as the *condition* of that same transition in Stateflow, and an effect is added to the transition if it is specified in SysML with an *opaque behaviour*. Two types of transition events are supported by the transformation: *relative time events*, which trigger after a specified time interval; or *signal events*, which are mapped to *messages* in Stateflow. These *messages* are defined as inputs of the Stateflow block and they can only be sent by the *Monitoring and Control* block (this topic is further discussed later in this section). In the example shown in Figure 3.13, the Stateflow block *STM1* has an input that corresponds to the message *SignalEvent1*, which is used in the definition of one or more internal transitions.

*Relative time events*, *guards*, and *effects* of transitions may all be specified using time-independent parameters that are defined as *value properties* in the model. An example of this is available in Figure 4.8, in which the parameter *PointThresh* is used in the definition of two transitions. In this case, the MATLAB program can identify the use of these parameters and replace them with the value that is

defined in the table shown in Figure 3.10.

Another possibility is that the *guard* or *effect* of a transition uses a parameter that specifies the active state of another state machine. An example of this is available in Figure 4.8, in which the variable *scMode* is used in the definition of two transitions. In this case, the MATLAB program creates an additional input *data object* in the Stateflow block, which is connected to a *from flag* that accesses the corresponding globally accessible variable. Figure 3.13 shows an example in which a transition in *STM1* uses the variable *activeState\_STM2*, which holds the active state of a state machine named *STM2*.

Finally, the *guard* and *effect* of a transition may also use a time-dependent parameter that is defined as a *value property* of the block that owns the state machine in question. This *value property* must be connected to a *port* of one of the block's composing elements. An example of this is available in Figure 4.8, in which the parameter *angRate*, representing the instantaneous angular rate of the spacecraft, is used in the definition of two transitions. An input *data object* is defined for this parameter and the same connection modelled in SysML is replicated in the Simulink model by the MATLAB program. In Figure 3.13, the parameter *timeVariantParameter1*, which is associated to an output of a composing element, is used in the specification of one or more transitions in *STM1*.

### Using Simulink Semantics in SysML

In some cases, the SysML semantics are not able to precisely represent some aspects of the system that are algebraic in nature. To solve this problem, Simulink semantics may be imported into SysML using the *stereotypes* discussed in Section 2.4, *«SimulinkBlock»* and *«SimulinkParameter»*. In this work, there was the need to represent the sum of vectors and scalars in an *ibd*, and this was accomplished using a *Sum* block from Simulink. Also, the multiplexing of three scalar components into a single vector and the inverse demultiplexing were easily represented using the Simulink native blocks *Mux* and *Demux*. When a Simulink model is generated with the *Simulink Export* function, these blocks that were imported to SysML are automatically replaced by the corresponding native Simulink blocks.

### Specifying the *Monitoring and Control* block

The *Monitoring and Control* block, part of the FES structure as shown in Figure 2.5, is defined in [60] as consisting of scenario definition functions and post-processing functions. With this approach, this element of the *analysis context* owns *ports* that are connected to the *Mission Context* block. These *ports* are internally connected to blocks that represent the processing of time-dependent parameters. This processing results in the generation of error indices that are necessary for posterior derivation of TPM values. This process is discussed in Appendix B. The elements that process time-dependent parameters are defined as components in SysML and are specified mathematically with *Subsystem* blocks in Simulink.

The *Monitoring and Control* block in Simulink also sends *messages* that correspond to *signal events* in SysML. These *messages* are necessary to trigger transitions in the Stateflow blocks at specific instants, according to the defined simulation scenario. The instants in which the *messages* are sent are

specified in SysML with an *interaction*, which defines the specific scenario that is simulated in Simulink.

### Configuring and Executing the Simulation Model

The properties of the Simulink model, such as *Stop Time*, *Start Time*, or *Step Size*, can also be specified in SysML. For each of these, a *value property* is created and attributed to the *analysis context*. The values for these parameters are defined in the table shown in Figure 3.10, and they are attributed to the corresponding Simulink model properties by the MATLAB program.

There are two properties of the simulation model that are specifically relevant in this approach: the *StartFCN* property, specifying MATLAB functions that are run before the simulation; and the *StopFCN* property, specifying MATLAB functions that are run after the simulation. The first is needed to specify the MATLAB functions required for the setup of the various mathematical models. The second is needed to specify the MATLAB functions that are executed to derive TPM values from the generated error indices. These function calls are specified in SysML, including their input parameters, which may use *value properties* specified in the model in their definition. The MATLAB program processes these function calls, replacing *value property* names with the corresponding values defined in SysML.

The MATLAB functions that derive TPM values from the simulation results are simple and tailored to each specific TPM. In addition to deriving these values, these functions write the values in an Excel spreadsheet that is synchronised with a table in SysML. This way, the MATLAB functions can overwrite the values defined for the TPMs inside the SysML model.

## 3.2 Model Setup

Model organization impacts keys aspects of the development process such as reuse, access control, and model navigation [14]. The model developed in this work is organized in a similar way to the model developed in [21] (see Figure 3.15). The model hierarchy replicates the system hierarchy, and at each architecture level, a set of *packages* are created to store different types of elements (e.g., requirements, structural modelling, and behavioural modelling are stored in distinct *packages* at each level).

While a certain SOI is being designed, different levels of abstraction of the system are specified, namely the functional, logical, and physical architectures. The specification of each of these is stored in different *packages*, as shown in Figure 3.15. In this work, this only applies at the subsystem-level, since the system-level has already achieved a stable configuration. Inside each of these *packages*, the division into element types is also implemented.

Some *packages* and element types are only present at the mission-level of the model structure, such as analysis elements, and supporting elements. The *Analysis package* contains the model elements that are created to support the *Optimize and Evaluate Alternatives* activity, and the supporting elements include definitions of general *value types*, *port types*, and signals. There are also different packages for the spacecraft and ground station, since these are defined separately.

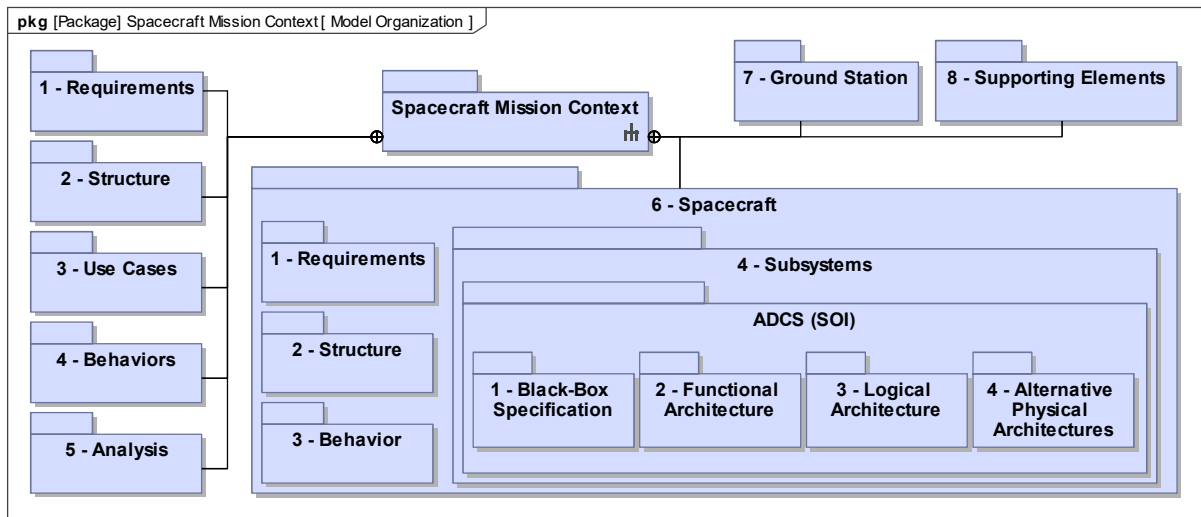


Figure 3.15: Model Organization.

### 3.3 Methodology Validation

The methodology discussed in this chapter is validated by being applied to the design of an ADCS of a 1U CubeSat (composed of one unit). A CubeSat is a type of standardized nanosatellite composed of one or more units, which are shaped as a 10 cm cube with a mass of up to 1.33 kg [83]. This design scenario follows the subsystem design process definition presented in Figure 2.8. The ADCS is an appropriate subsystem for the validation of an MBSE design approach because its design is multidisciplinary and requires the use of simulation in early phases of the design, for performance evaluation.

The mission objective for this CubeSat concerns the observation of the Earth. The system is equipped with a payload that must be oriented with nadir to perform its sensing activities. This subsystem design scenario is not developed in the context of a real CubeSat mission, and for that reason both the mission and the spacecraft are not fully specified. Instead, these have been specified in terms of the aspects that are most relevant for the design of the ADCS. The ISTSat-1 CubeSat mission [84], currently being developed in Instituto Superior Técnico under ESA Education's *Fly Your Satellite!* program, is used in this work as a basis for defining realistic characteristics of the system and mission.

The main objective of applying the developed MBSE methodology to this design scenario is not the definition of the best design for the ADCS of this CubeSat. Instead, we wish to validate the methodology with a design scenario that has an adequate level of complexity. If the complexity level is too low, the methodology validation is undermined, and if it is too high, the scope of the work increases unnecessarily and the proper understanding of the methodology is compromised. The ISTSat-1 ADCS design [85] will be used as a reference for the definition of an adequate level of complexity.

In the next chapter, the design inputs are first defined and then the methodology is applied to this design scenario. The purpose of this methodology application is both to provide a more detailed description and demonstration of the methodology and to validate the methodology as a whole. Apart from being a demonstration of the specific MBSE methodology proposed in this work, this can also be seen as a demonstration of the capabilities of an MBSE approach in general.

## Chapter 4

# Methodology Demonstration

### 4.1 Design Process Inputs

In this section we specify the design process inputs for the specific design scenario implemented in this work, namely the design of an ADCS of a 1U CubeSat. The specification of these inputs, which are identified in Figure 2.8, provides the necessary context for the design of the subsystem in question.

#### 4.1.1 Mission-level inputs

The definition of the mission architecture, shown in Figure 4.1, is essential to provide context for the design of the system. The block named *Mission Context* specifies the integration of the mission enterprise with other external entities relevant to the mission, such as the environment. The mission enterprise is composed of the systems which comprise the solution to the mission objectives and whose definition is under the control of the development team, which in this case are the spacecraft and the ground station. The CubeSat is intended to be launched as a secondary payload, and for that reason, the launcher is not under the control of the development team, hence not being considered as a part of the mission enterprise. In this work, we define the spacecraft environment (or just environment) to be composed of the spacecraft translational and rotational dynamics, the interactions between the spacecraft and other astronomical bodies, and the orbits of relevant astronomical bodies. The environment is modelled alongside the systems so that the interactions with it may be better understood and specified.

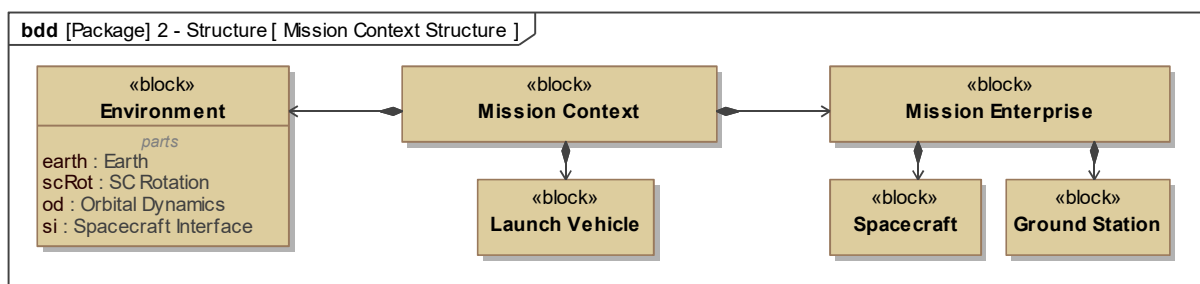


Figure 4.1: Mission architecture.

Some parameters relevant to the simulation of the ADCS performance are attributes of the mission as a whole, and these are consequently defined as *value properties* of the *Mission Enterprise* block. Two of these, the orientation of the angular velocity and the initial attitude at the instant of launcher separation, are assumed to be unpredictable. In simulation, a worst-case scenario must be defined for these, as discussed in Appendix B. The other relevant parameters, namely the instant of separation, the maximum expected angular rate at separation, and the entry orbit, are considered to have been defined in a previous design phase, and their definition is shown in Table 4.1. The instant of separation was defined arbitrarily and the other two parameters were defined as equal to those of the ISTSat-1 mission.

| Attribute                          |                                    | Value                        |
|------------------------------------|------------------------------------|------------------------------|
| Instant of launcher separation     |                                    | 1 January 2022, 12:00:00 UTC |
| Maximum angular rate at separation |                                    | 100 deg/s                    |
| Entry orbit                        | semi-major axis ( $a$ )            | 6871 km                      |
|                                    | eccentricity ( $e$ )               | 0                            |
|                                    | inclination ( $i$ )                | 97.86 deg                    |
|                                    | RAAN ( $\Omega$ )                  | 79.1815 deg                  |
|                                    | Argument at perigee ( $\omega_p$ ) | 0 deg                        |
|                                    | Mean anomaly ( $M$ )               | 0 deg                        |

Table 4.1: Definition of mission attributes. The entry orbit is the same as for the ISTSat-1: Sun-Synchronous Orbit (SSO) with LTAN at 22:30.

#### 4.1.2 System-level inputs

The system white box specification provides context for the design of all the subsystems. It consists of the identification of all the subsystems and the interactions between these. A **bdd** is shown in Figure 4.2 representing the decomposition of the system into subsystems (based on the one developed in [21]).

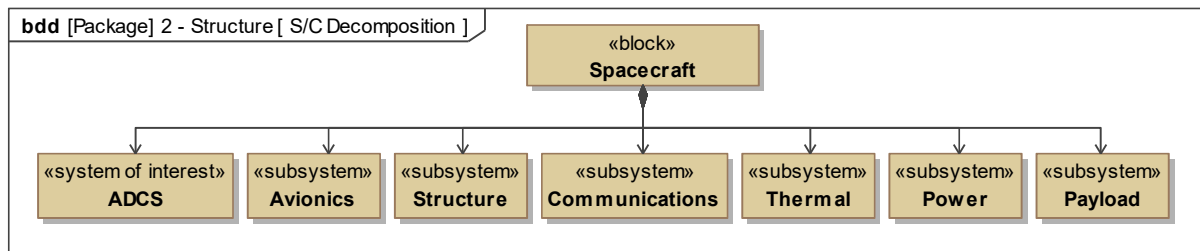


Figure 4.2: System decomposition.

The part of the mission and system context that is relevant for the ADCS design is shown in Figure 4.3. The ADCS interacts with the environment by measuring the angular velocity of the spacecraft and a set of known vector quantities, while also producing external torques and using momentum accumulator devices to control the spacecraft attitude. An ADCS in general interacts with many of the other subsystems [22], both physically and in terms of design definition. However, modelling all these interactions would result in unnecessary modelling effort. Therefore, only the interaction with the avionics subsystem



is modelled in this work, so that the methodology is properly validated in terms of the definition of interfaces between subsystems. In Figure 4.3, we can see that the avionics subsystem provides an electrical supply to all the hardware components used in the ADCS, while also receiving a periodic data report.

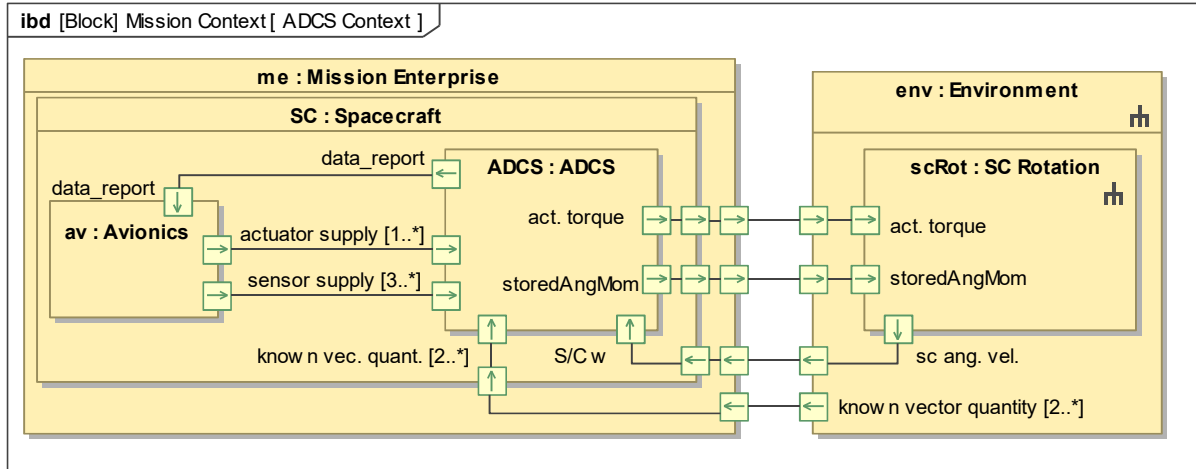


Figure 4.3: Mission- and system-level context of the ADCS.

Another of the inputs to the design process is the specification of the system behaviour, which was modelled with a state machine, as shown in Figure 4.4 (based on the model in [21]). Regarding SysML nomenclature, the yellow-filled boxes represent states, the arrows represent transitions, and the black-filled circles represent *initial pseudostates* (where the execution starts when the state machine or state becomes active). Some transitions are triggered by the reception of signals from a third-party (e.g., the reception of a *Reboot Cmd* signal triggers a transition from the *on* state to the *rebooting* state).

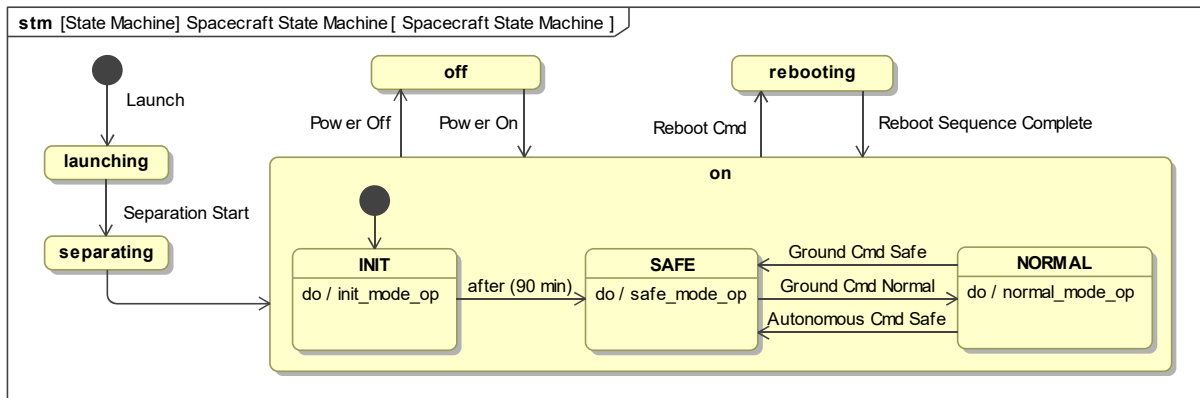


Figure 4.4: Spacecraft state machine behaviour specification.

In Figure 4.4, the composite state *on* contains a set of substates representing the operational modes of the spacecraft, which are the same as the ones defined in the ISTSat-1 design. After separation, the spacecraft enters in the *init* mode, in which the communications subsystem is disabled as a safety measure. The *normal* mode represents the normal operation of the system, and the *safe* mode represents a safe state of the system, in which all non-essential functions are shut down. The state machine is only shown with a composite state to facilitate the understanding of the reader, given that this is not

supported by the model transformation approach described in Section 3.1.3. In the developed model, the composite state is instead defined with a second state machine that specifies its behaviour.

Each of the substates of the state *on* owns a *do* behaviour, i.e., a behaviour that executes continuously until it terminates on its own or an event like a transition interrupts it. These behaviours are specified with *activities*, and they are a part of the classifier behaviour of the *Spacecraft* block.

With the decomposition of the system into subsystems, parts of the system behaviour are allocated to each subsystem. Regarding the part that is allocated to the ADCS: (i) in *init* mode, the ADCS must detumble the spacecraft, reducing its angular speed below a certain threshold; (ii) in *normal* mode, it must align the payload with the nadir using all the available algorithms and hardware; and (iii) in *safe* mode, it must align the payload with the nadir but while consuming a much lower average power (at the cost of decreased accuracy). More details are included in the definition of the subsystem requirements.

There are also some attributes of the system that are relevant for the simulation of ADCS performance. These were modelled as *value properties* of the *Spacecraft* block, and their values are shown in Table 4.2. All these values are based on the specification of the ISTSat-1.

| Attribute                                     | Value                                                                                                                                            |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Mass ( $m$ )                                  | 1.33 kg                                                                                                                                          |
| Coefficient of drag ( $C_D$ )                 | 2.2                                                                                                                                              |
| Coefficient of reflectivity ( $C_r$ )         | 1.8                                                                                                                                              |
| Drag reference area ( $S_{\text{drag}}$ )     | 0.01 m <sup>2</sup>                                                                                                                              |
| SRP reference area ( $S_{\text{SRP}}$ )       | 0.01 m <sup>2</sup>                                                                                                                              |
| Moment of inertia ( $I_s$ )                   | $\begin{bmatrix} 1.6195 & -0.0174 & 0.0113 \\ -0.0174 & 1.7603 & 0.0036 \\ 0.0113 & 0.0036 & 1.8415 \end{bmatrix} \times 10^{-3} \text{ kg m}^2$ |
| Undesired magnetic dipole magnitude ( $m_u$ ) | 0.01 A m <sup>2</sup>                                                                                                                            |

Table 4.2: Definition of relevant system attributes.

### 4.1.3 Subsystem-level inputs

The subsystem requirements are the major driver of the complete design process. Their definition is part of the *Manage Requirements Traceability* activity, which is applied across all the levels of the system architecture. The part of the *specification tree* that is relevant to the ADCS is shown in Figure 4.5, in which the three requirements represent containers for other requirements, at different levels of the architecture. The ADCS requirements are stored in the lower-level container, and they are derived from higher-level requirements or identified during the system- and subsystem-level design definition.

The *ADCS Requirements* container is decomposed into four other containers, representing: (i) functional requirements, which describe the subsystem functions qualitatively; (ii) operational requirements, which define conditions or properties required during system operation; (iii) performance requirements, which define quantitatively how well a function is to be performed; and (iv) design constraints, which consist of design decisions made in previous phases of the project. The division of the requirements into categories helps keep the model organized.

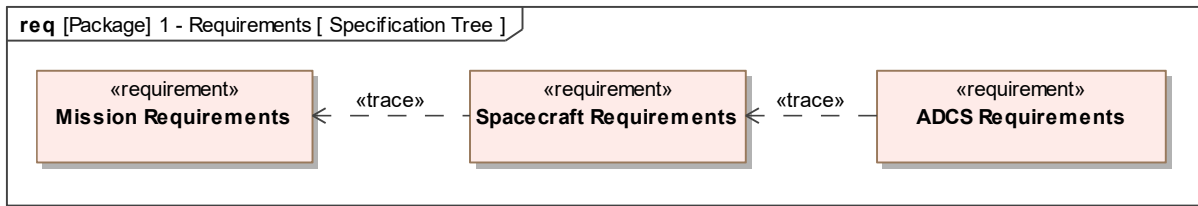


Figure 4.5: *Specification tree* (only the part relevant to the ADCS).

The definition of the ADCS requirements was based on the ECSS [86], so that these are similar to what would be defined in a real design scenario developed in an industry setting. The defined ADCS requirements are summarized below (a complete list can be consulted in Appendix B):

**Functional Requirements** A mapping between ADCS modes and spacecraft modes must be clearly defined (Req. 7.1.1.3) and mode transitions shall be triggered both by ground request and autonomously (Req. 7.1.1.2). A special *Safe Pointing* mode must be activated when the spacecraft is in *safe* mode. In this mode, magnetic torquers are the only actuator that can be used (Req. 7.1.1.1), because these consume a low average power and have been proven to be reliable. In addition, the safe transition from the initial conditions after launcher separation to the final mission pointing must be ensured by the ADCS, using a detumbling algorithm (Req. 7.1.2). The ADCS shall also deliver attitude and orbit related information to the OBC software, in the form of a data report (Req. 7.1.3).

**Operational Requirements** In-flight updates of ADCS design parameters must be provided (Req. 7.2.1) and all the parameter update periods must be greater than the maximum period without contact with the ground station (Req. 7.2.2). The ADCS shall also ensure continuous reaction wheel off-loading (Req. 7.2.6). Other operational requirements define conditions relative to the detumbling (Req. 7.2.3 and Req. 7.2.4) and minimum ADCS coverage (Req. 7.2.5).

**Performance Requirements** The nomenclature and mathematics used in the definition and verification of performance requirements are reviewed in Appendix B. Performance requirements were defined concerning the absolute pointing performance, both in *safe* mode and *normal* mode (Req. 7.3.1 and Req. 7.3.2, respectively). Another requirement concerns the absolute attitude knowledge performance, which relates to the accuracy of the attitude determination (Req. 7.3.3). The accuracy of the on-board spacecraft position estimation is addressed by another performance requirement (Req. 7.3.4).

The development of budgets at the system-level (e.g., mass or power) usually results in the definition of subsystem requirements. These budgets are divided across the multiple subsystems according to estimates that are developed, and then they are updated at each step of the system development process. In this design scenario, mass and average power consumption are constrained as part of budgets that are defined at the system-level (Req. 7.1.4 and Req. 7.2.7, respectively).

As mentioned above, some subsystem requirements are derived from other higher-level requirements. Representative examples of this practice were modelled to show how MBSE supports the task

of deriving requirements at successive levels of the architecture. This process is illustrated in Figure 4.6 (the yellow note-shaped boxes outline the rationale of a certain requirement relationship).

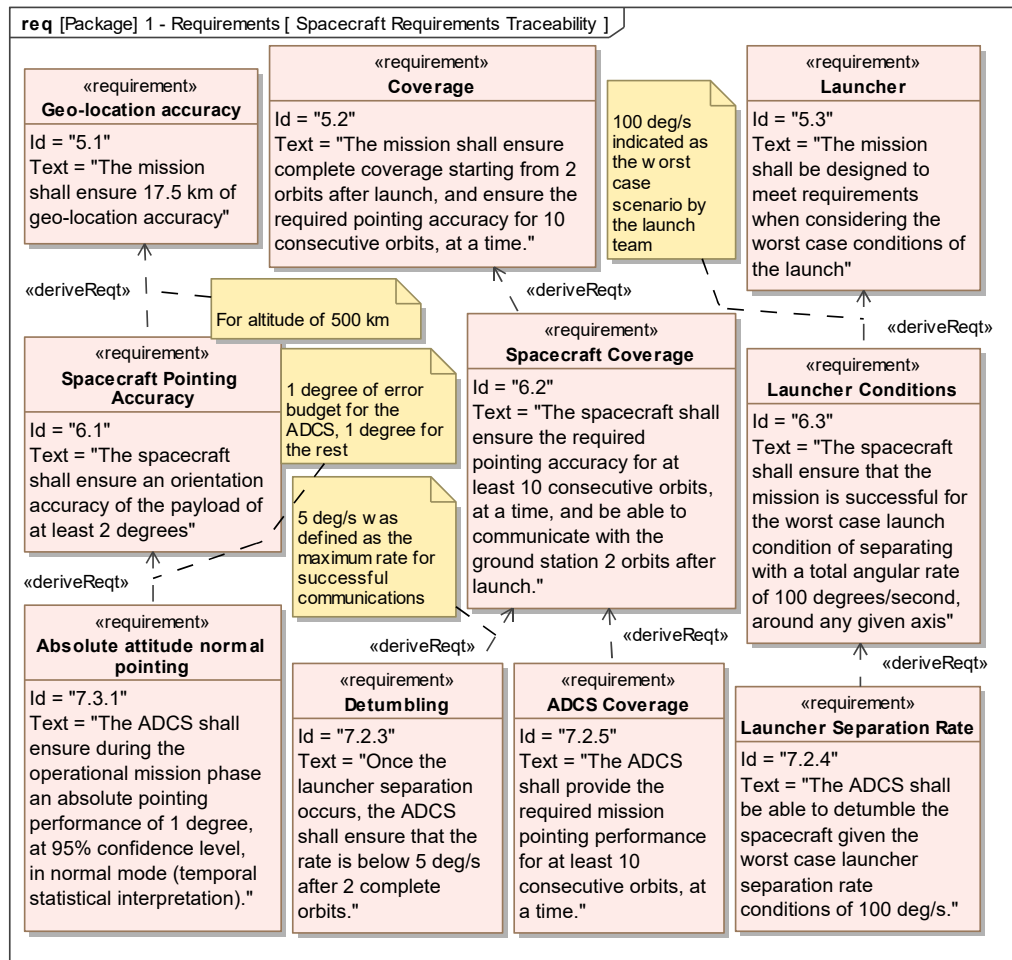


Figure 4.6: Example of traceability between mission-level, system-level, and ADCS requirements.

The requirement concerning absolute attitude performance in *normal* mode was derived from a system-level requirement regarding the accuracy of the payload orientation, which in turn was derived from a mission requirement concerning geo-location accuracy. The geo-location error was translated into an angular orientation error by considering the altitude of the orbit. Then, the payload orientation error was divided into a budget for the ADCS and a budget for the other subsystems (e.g., the structure).

The same type of process was applied to derive both an ADCS coverage requirement and a detumbling requirement from a mission-level coverage requirement. The maximum expected launcher separation angular rate was defined in a previous design phase, taking into consideration the launching conditions. This resulted in the definition of requirements for every subsystem that may be influenced by this mission attribute, such as the ADCS.

When a requirement is changed at any level of the architecture, or the verification of a lower-level requirements fails, automatic impact analysis may be performed, enabled by the traceability introduced by these requirement relationships. This process is important to ensure that the verification of lower-level requirements leads to the verification of higher-level requirements, but more importantly, to the validation

of the system design against stakeholder needs.

A few design constraints have also been defined for this design scenario, representing design decisions which took place in the conceptual design phase. These enable a progressive reduction of scope of the design activities, as the subsystem becomes better defined. In this case, the constraints help limit the scope of the design to an appropriate degree for the application of the proposed methodology.

The first constraint imposes that the ADCS software be allocated to the OBC hardware (Req. 7.4.1), such that the ADCS does not include a dedicated processing unit. Also, the ADCS design shall: (i) be based on active control techniques (Req. 7.4.2); (ii) include an attitude filtering algorithm (Req. 7.4.3), and (iii) ensure on-board estimation of the spacecraft position in the ECI frame (Req. 7.4.5). Finally, the design space of the ADCS was also constrained in terms of hardware selection (Req. 7.4.4).

All the hardware used in the ADCS of the ISTSat-1 is also used in this design. The ISTSat-1 is equipped with one gyroscope, one magnetometer, one CSS in each face of the spacecraft (except  $Z^-$ ), and three magnetic torquers mounted orthogonally. The CSSs and magnetic torquers are integrated into the solar panels. These components constitute the initial design configuration of the ADCS. The magnetometer is inexpensive, reliable, and small, and it is required for the computation of the magnetic torquers' commanded dipole. The gyroscope is essential to the attitude filtering and the CSSs are already integrated into the solar panels placed in each face of the spacecraft (except  $Z^-$ ).

Aside from these, only two other types of components may be used. The CSSs provide Sun vector estimates only out of the eclipse, and which can be quite inaccurate due to the influence of the Earth albedo. To address this limitation, the addition of one Earth sensor mounted on the face  $Z^-$  is considered. Also, if a better attitude control performance is required at any point, three reaction wheels mounted orthogonally are added to the ADCS design. All of these constraints to the hardware selection are reflected in Req. 7.4.4.

The final input to the design process is the black box specification of the subsystem, which is shown in Figure 4.7. The interface of the subsystem is defined by a set of *ports* of different types and with different multiplicities. This interface definition is consistent with what is represented in Figure 4.3. A set of quantitative attributes of the system are modelled as *value properties*, whose definition is mostly based on the analysis of the subsystem requirements. The *«tpm» stereotype* identifies the *value properties* that represent TPMs, which are attributes that are directly targeted by performance requirements. The mathematical definition of a TPM is presented in Appendix B. A set of *operations* and *receptions* are also defined, which are identified throughout the process of defining the behaviour of the system and other subsystems.

Each of the TPMs identified in Figure 4.7 is associated to exactly one performance requirement. The *value properties* named *stmPeriod*, *DetThresh*, and *PointThresh* are time-independent parameters that are used in the definition of the ADCS state machine (presented in Section 4.2.1). Also used in the definition of this behaviour are the *value properties* *angRate*, *autoTransEnabled*, and *adcsMode*, which represent time-dependent parameters. The rest of the *value properties* all represent quantitative attributes of the subsystem that are constrained by either functional or operational requirements.

In terms of *operations*, the *forceModeTransitions()* ensures that mode transitions in the ADCS can be

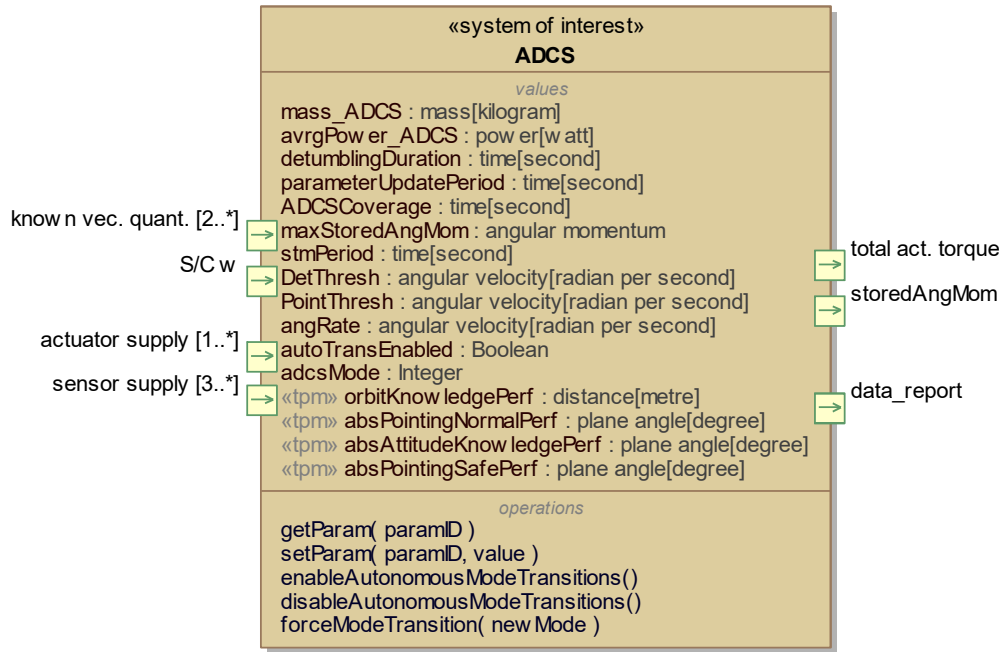


Figure 4.7: ADCS black box specification.

externally forced (Req. 7.1.1.2). The *operations* that enable and disable autonomous mode transitions are defined to satisfy Req. 7.1.1.4, and *getParam()* and *setParam()* represent common functions of software components, which also satisfy Req. 7.2.1.

## 4.2 Subsystem Design

This section concerns the design definition of the ADCS, which is developed according to the methodology described in Chapter 3. First, its behaviour is defined with a state machine, then the functional and logical architectures are developed. After this, the physical layer is specified, in which two candidate architectures are defined: (i) an ADCS design including reaction wheels and magnetic torquers; and (ii) an ADCS design including only magnetic torquers.

All the activities above concern the specification of the subsystem design. After this specification is completed, simulation must be performed, for requirements verification and to support the comparison between the two alternative designs mentioned above. Before simulation can be executed, the environment must be specified in SysML, in terms of its internal structure and the black box representations of its components. Then the simulation itself can be specified through the definition of the *analysis context* and the *Monitoring and Control* block, as discussed in Chapter 3.

The generated simulation model is not shown in any figures, since its structure does not differ from what is shown in the **ibd** diagrams across this section. Instead, simulation results are presented, including the values obtained for the TPMs and plots of some of the error indices computed. The simulation scenario presented in this section was defined with the purpose of enabling the verification of all the requirements it concerns (see Figure 4.22).

## 4.2.1 Subsystem State Machine Definition

The definition of the subsystem design begins with the specification of its behaviour, using a state machine (see Figure 3.1). This behaviour specification requires making design decisions, even though a considerable part of it is derived directly from the design process inputs. A state machine representing a subsystem's behaviour may become too complicated to represent in one single diagram. Instead, we created a set of simpler diagrams, which show different views of the same behaviour. Changes to the behaviour specification may be applied by editing any of these and consistency is automatically ensured.

The ADCS has three modes of operation: (i) *normal pointing*, representing the regular operation of the subsystem; (ii) *safe pointing*, to satisfy Req. 7.1.1.1; and (iii) *detumbling*, to satisfy Req. 7.1.2. The transitions between the three ADCS modes are shown in Figures 4.8 to 4.10. In these, *angRate* is a *value property* that represents the instantaneous angular rate of the spacecraft, while *scMode* is the variable that holds the active substate of the composite state *on*, shown in Figure 4.4. Regarding SysML nomenclature, expressions inside square brackets represent the *guard* of a transition, i.e., the condition that must be verified for the transition to occur. The small black-filled circles are *junction pseudostates*, which specify compound transitions. For each *junction pseudostate*, exactly one exit transition must be valid at a time (the *guard* “else” may be used to ensure this).

While in any mode, a possible transition is periodically evaluated (with a period of *stmPeriod*), which may only occur if the autonomous transitions are enabled (or *autoTransEnabled* == *true*), satisfying Req. 7.1.1.4. If this condition is met, the logic of the transitions originated in the *detumbling* mode is the following (see Figure 4.8): in the spacecraft mode *normal*, if the angular rate is above a certain threshold — *PointThresh* — the *detumbling* mode remains active, otherwise a transition to *normal pointing* occurs; in the spacecraft mode *safe*, the exact same logic applies but with *safe pointing* replacing *normal pointing*; in *init* mode, the *detumbling* state must be active. With this logic, in the case of a failure occurring in the pointing modes, the angular rate is kept below an acceptable threshold for communications.

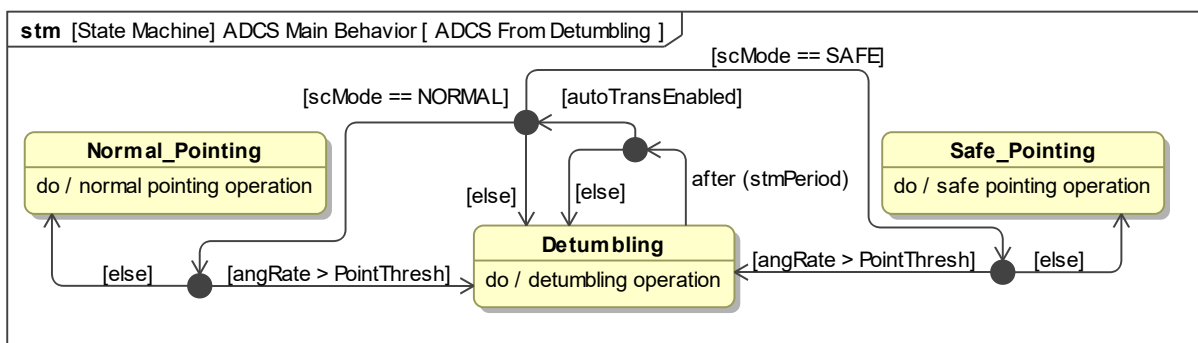


Figure 4.8: Representation of the autonomous transitions originated in the *detumbling* mode.

Regarding, the transitions originated in the other modes, the logic is very similar. The differences reside in the fact that the spacecraft mode *init* must not be active (see Figure 4.4) and that the angular rate threshold is different (*DetThresh*). The angular rate threshold defined for transitions originated in the *detumbling* mode is lower than the threshold applicable for transitions from the other modes, in order to avoid rapid sequences of back-and-forth transitions.

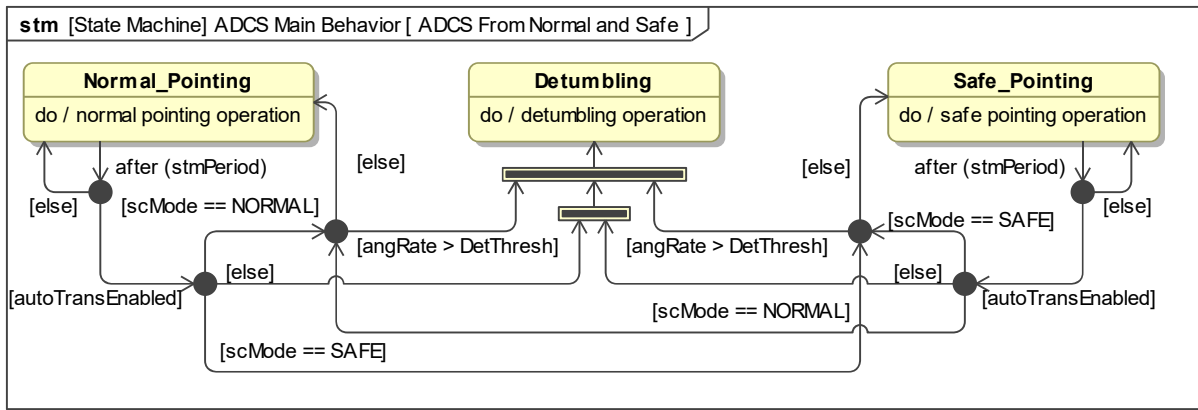


Figure 4.9: Representation of the autonomous transitions originated in the pointing modes.

The final **stm** diagram, shown in Figure 4.10, represents mode transitions forced by ground request, which satisfy Req. 7.1.1.2. These can be triggered at any moment, by invoking the operation *forceModeTransition()*, which receives as an input the mode that should be activated. This input parameter — *newMode* — can be equal to *NP*, *SP*, or *DET* which correspond to *normal pointing*, *safe pointing*, and *detumbling*, respectively. If the operator wants the forced mode to remain active, the autonomous transitions may be disabled. Also represented in the **stm** diagram is the initialization of the state machine, which always starts in the *detumbling* mode.

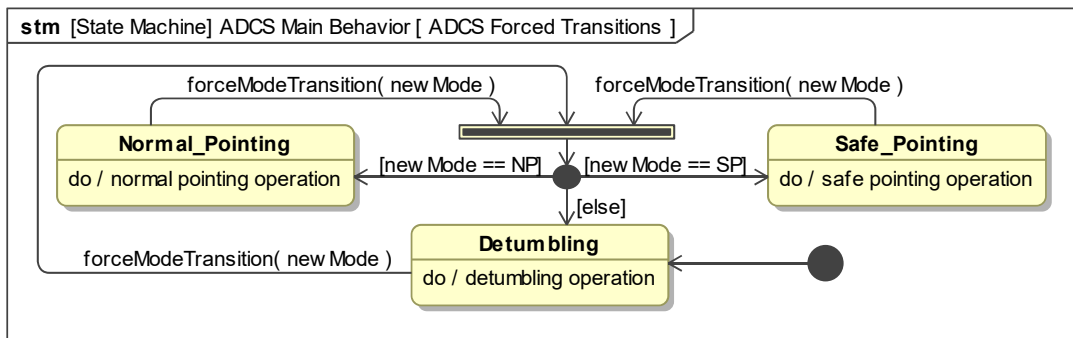


Figure 4.10: Representation of the forced transitions and state machine initialization.

In a simulation of the ADCS performance, this state machine must be included in the simulation model, because the behaviour of the ADCS and its components depends completely on the ADCS mode that is active in each instant. This can be accomplished with the model transformation framework described in Section 3.1.3, as it is capable of transforming SysML state machines into Stateflow blocks.

## 4.2.2 Functional Architecture Definition

The functional architecture definition, which follows the process shown in Figure 3.3, starts with the decomposition of the subsystem into components. In this work, architecture decompositions introduce two levels in the hierarchy, one used to divide software and hardware into different *blocks*, and the other used to define the decomposition of the software and hardware. Yet, these two levels are equivalent to only one architecture level (both hierarchy levels represent the system definition at the component-level).



The functional decomposition of the subsystem is shown in Figure 4.11. Following the ADCS structure represented in Figure 2.9, different entities are defined for the attitude determination and attitude guidance and control (these last two are joined together, as mentioned in Section 2.6). The ADCS technology review presented in Section 2.6 divides sensors and actuators into types according to their function, and this type of decomposition is replicated in the definition of this functional architecture. Actuators are divided into momentum accumulators and torque actuators, while sensors are divided according to what they measure, namely the spacecraft angular velocity or a known vector quantity.

Attitude determination based on filtering (as imposed by the design constraint 7.4.3) requires both types of sensors as well as the estimation of the known vector quantities represented in the inertial frame. The knowledge of the spacecraft position in the inertial frame is also required, both as an input for some of the algorithms and to satisfy a set of different requirements (e.g., Req. 7.1.3). It must result from the execution of an estimation algorithm, as stated in the design constraint 7.4.5.

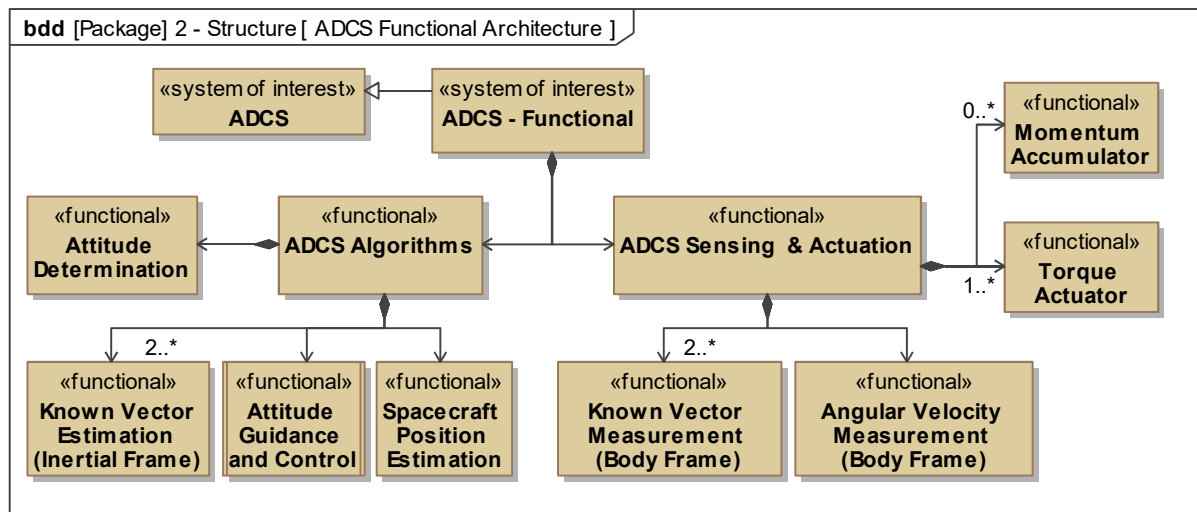


Figure 4.11: Functional decomposition of the subsystem.

In SysML, this type of decomposition may also specify multiplicities for the components. In this case, the specified multiplicities define that: (i) two or more vector quantities must be measured in the body frame and estimated in the inertial frame (to enable complete attitude determination); (ii) torque actuators must be included in the design and more than one may be used; and (iii) the inclusion of momentum accumulators is not mandatory, and their quantity is unrestricted.

After the decomposition is defined, interactions between the components must be modelled to realize each of the *activities* attributed to the subsystem. We focus on the *activities* attributed to each mode, identified as *do* behaviours of the states represented in Figures 4.8 to 4.10. For each *activity*, an **act** diagram is created. This type of diagram shows the definition of the *activity* in terms of a sequence of *actions*, and the *tokens* that are exchanged between the *actions* (in this case, signals). The *actions* are represented in the diagram as boxes with rounded corners, *token* exchanges are represented as arrows, and *parameter nodes*, which represent inputs and outputs of an *activity*, are represented as rectangular boxes placed over the limits of the diagram frame. The allocation of each of the *actions* to a specific component is also specified in **act** diagrams, with the use of *swimlanes* (structures similar to a table with

multiple columns), which are structured according to the hierarchy of the subsystem architecture.

The subsystem behaviour in the *detumbling* mode is represented in Figure 4.12. It consists of an algorithm that uses the measurement of angular velocity to compute an actuation torque to be produced by the torque actuator. Also, a data report is generated, containing information relative to the measurement of angular velocity, position and velocity estimation, and eclipse status, as specified in Req. 7.1.3 (only the attitude is not included, since it is not estimated in this mode).

The behaviour of the subsystem in the *normal pointing* mode is represented in Figure 4.13. In this case, the *swimlanes* are both horizontal and vertical to ensure that the diagram fits in an A4 page. As we can see, the attitude determination algorithm uses both the measurement of angular velocity and the measurement and estimation of known vector quantities in the body and inertial frames, respectively. In some cases, the estimation of these vector quantities requires the knowledge of the spacecraft position. In turn, the attitude guidance and control algorithm computes the commanded torque, relying on the estimation of attitude and on the knowledge of the spacecraft position (for guidance). It also uses an estimate of the angular velocity corrected by the filter (with the estimated bias removed). If momentum accumulators are included in the design, these are used in this mode, otherwise torque actuators are used. Since these two options are both possible at this architecture-level, both are represented in the diagram. Finally, a data report conforming to Req. 7.1.3 is also generated as an output of the *activity*.

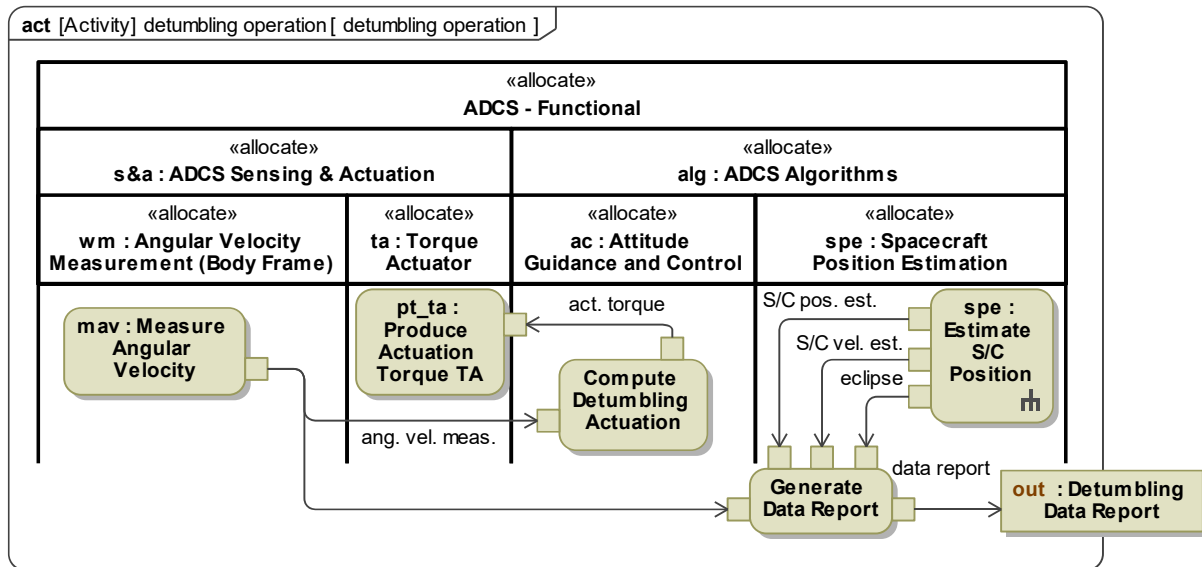


Figure 4.12: Interaction between functional components realizing the *detumbling operation activity*.

The behaviour of the subsystem in the *safe pointing* mode, represented in Figure 4.14, is very similar to the behaviour in the *normal pointing* mode. The main constraint is that the power consumption must be lower, which impedes the use of momentum accumulators and sensors with a high operating power. Because of this, different algorithms must be used for attitude determination and guidance/control.

As defined in the process shown in Figure 3.3, the specification of the subsystem's internal structure follows. This specification is based on the *activities* shown in Figures 4.12 to 4.14 and on the definition of the subsystem external interface. For this, an **ibd** is used, as shown in Figure 4.15. Connectors be-

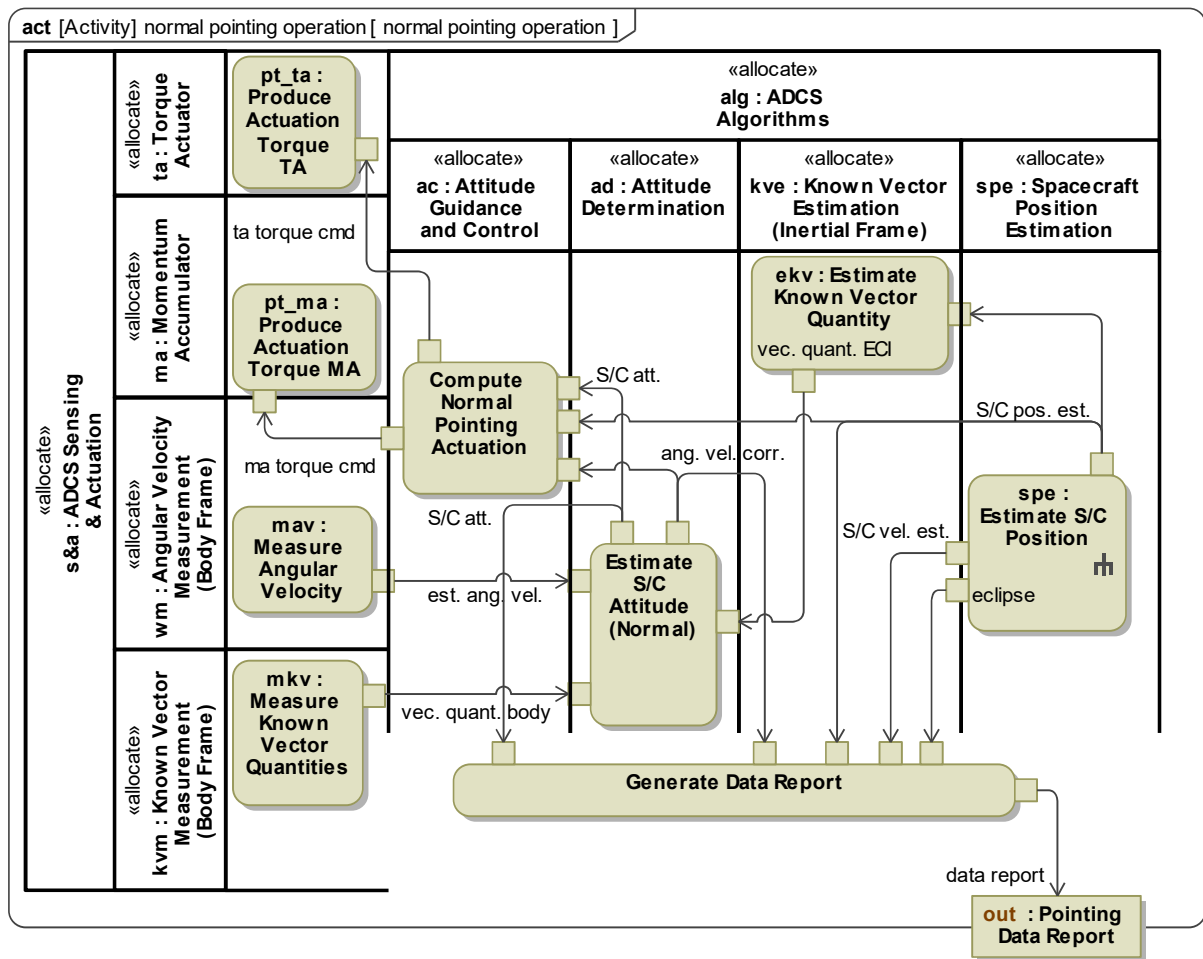


Figure 4.13: Interaction between functional components realizing the *normal pointing operation* activity.

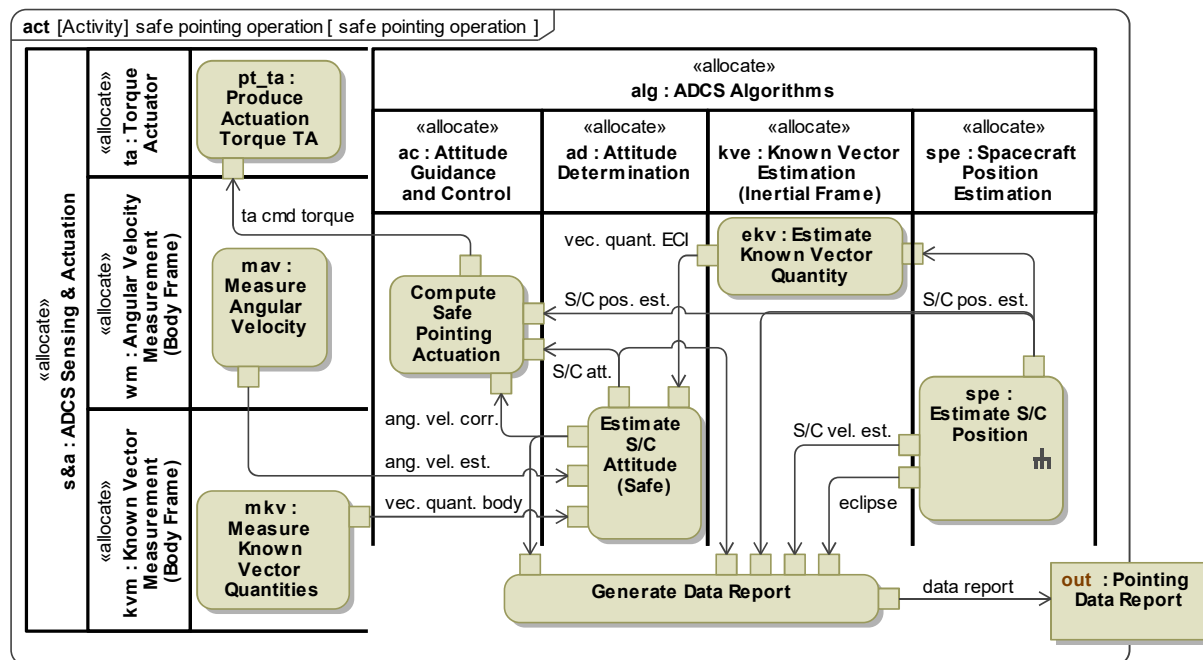


Figure 4.14: Interaction between functional components realizing the *safe pointing operation* activity.

tween any two components are defined to create communication channels for all the *tokens* exchanges specified in these *activities*. As explained in Section 3.1.3, these connectors must be routed through every level of the architecture, which results in an increase in the complexity of the diagrams and model.

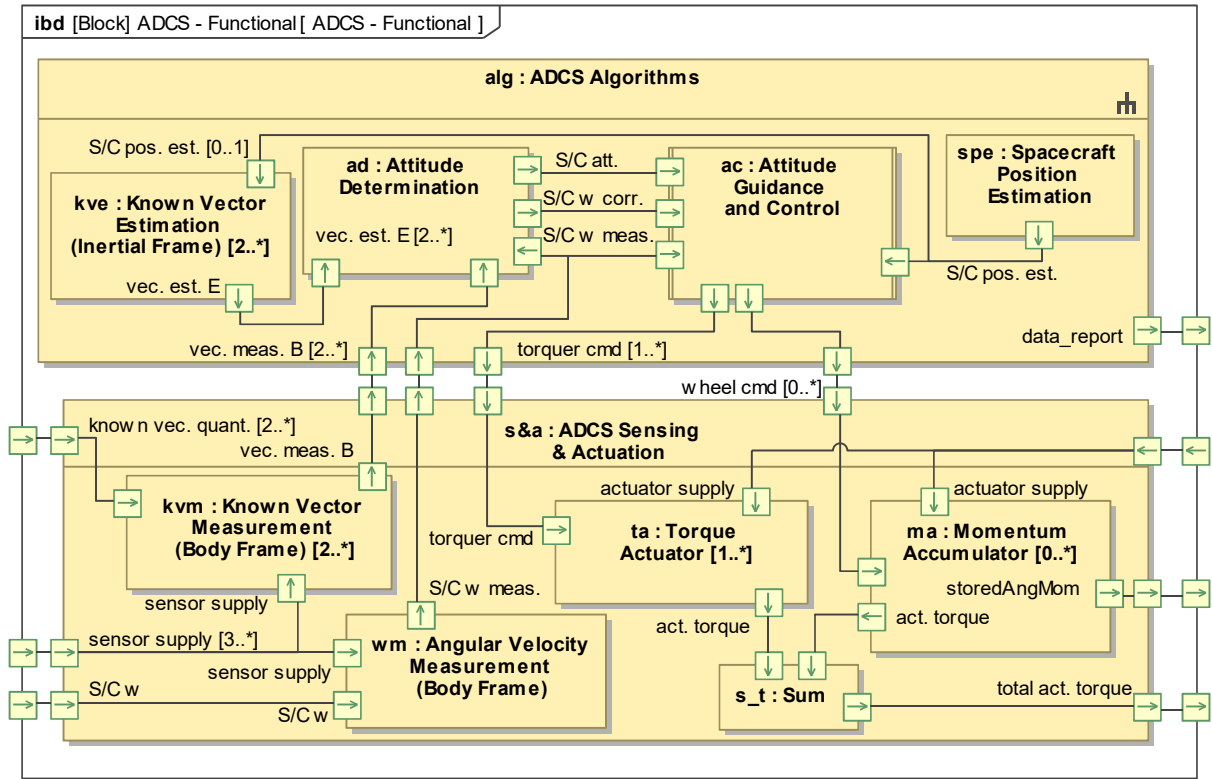


Figure 4.15: Internal structure of the ADCS functional architecture.

Special care must be taken to correctly define the multiplicities of the *ports*, and this correctness may be evaluated with the MBSE tool. For example, a *Momentum Accumulator* block always owns exactly one input *port* corresponding to a torque command (named “wheel cmd” in the diagram). However, the same is not true for the associated output *port* owned by the *Attitude Guidance and Control* block, which must have an unconstrained multiplicity (represented as “0..\*”), since this is the multiplicity of the *Momentum Accumulator* block.

The definition of the internal structure of the subsystem must also take into consideration how the external interface of the subsystem is connected to its composing elements. In Figure 4.15, the supply of each sensor and actuator is connected to the external interface, being further connected to the *Avionics* subsystem as shown in Figure 4.3. The same principle applies to the interface with the environment, e.g., the actuation torque produced by each actuator is connected to the external subsystem interface using an imported *Simulink* block that specifies a vectorial sum of the two torques.

### 4.2.3 Logical Architecture Design

The design of the logical architecture introduces the consideration of how the available technology may realize the functions defined for each functional component. It follows the process shown in Figure 3.4, in which the first step is the definition of the logical architecture of the subsystem. This is shown in the

**bdd** in Figure 4.16, together with the allocation of the functional components to logical components. Each of the logical components represented in the diagram is associated to a mathematical model that was developed inside the simulation environment, and which will be used in simulation.

Some functional components are allocated to more than one logical component because, at the logical level, components that were previously identical may become distinct. For example, the measurements of known vector quantities are realized at the logical level by three different types of sensors: a magnetometer, a set of five CSSs, and an Earth sensor.

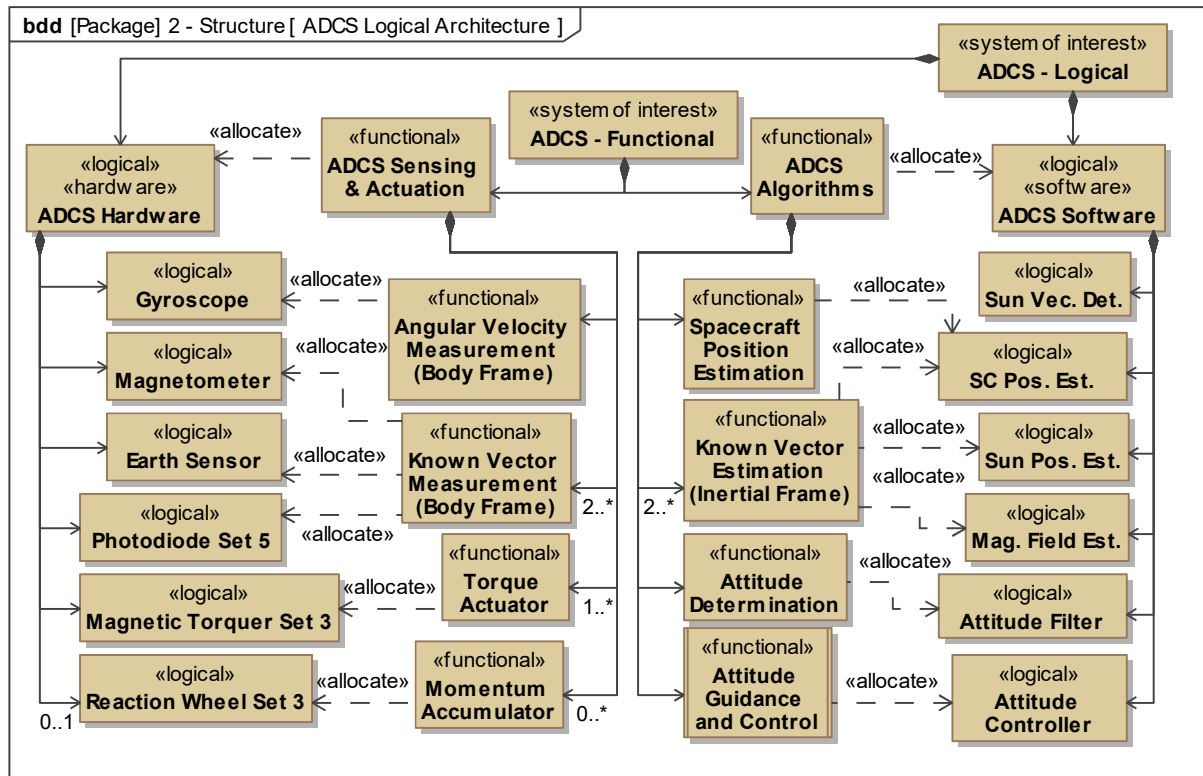


Figure 4.16: Logical architecture decomposition and allocation of functional to logical components.

Other hardware components include a gyroscope, three orthogonally mounted magnetic torquers, and three orthogonally mounted reaction wheels (which may or not be included in the design). The selection of these hardware components follows what is defined in the design constraint 7.4.4.

Some of the software components are also identified according to the selection of the hardware. Associated to each of the three sensors, estimates in the inertial frame of the local Earth magnetic field vector, the Sun position, and the spacecraft position are required (the nadir is derived from the last). Also, the need for a Sun vector determination algorithm is directly derived from the fact that photodiode readings must be combined and processed to obtain a Sun vector estimate. Two logical components representing an attitude filter and an attitude controller are also required, and their definition results from the direct refinement of the corresponding functional components.

The logical abstractions of all the components must be clearly defined at this point, so that these are clearly differentiated from their physical counterparts. At this layer, each hardware block represents a class of similar components, and not a specific procured or developed component. Also, the set of three

reaction wheels has a multiplicity between 0 and 1 (represented as “0..1”), since these will only be part of the design if the satisfaction of performance requirements depends on their inclusion. At the physical level, candidate architectures will be defined, and this multiplicity will be resolved to 0 or 1. The Earth sensor has an unspecified multiplicity (equivalent to 1) because at this point its inclusion in the design is decided. This is because the absolute pointing performance in *normal pointing* mode defined in Req. 7.3.1 is also required during the eclipse, and the Sun vector can only be determined out of eclipse.

In terms of software components, the blocks *SC Pos. Est.* and *Sun Pos. Est.* represent the estimation of the spacecraft and Sun positions, which are obtained via the numerical propagation of the orbit of the spacecraft and Earth, respectively. At this level, neither the specific integration method used nor the orbit perturbations that are considered in each case are defined. The algorithm used for the determination of the Sun vector is also not defined at this level, and the same applies to the selection of a specific model of the Earth magnetic field to be used in the *Mag. Field Est.* component. The attitude determination is performed with an attitude filter, but the specific algorithm used is only defined at the physical level. The *Attitude Controller* block encompasses all the control algorithms used in each mode, which are not fully specified at this level. However, it is defined that the pointing controllers are based on PD control techniques, since these are simple and have been shown to be effective in the design of the ISTSat-1.

The *activities* modelled at functional level are also modelled at this level, but in terms of an interaction between logical components. Apart from the difference in the level of abstraction, the process is identical.

The behaviour of the subsystem in the *detumbling* mode, represented in Figure 4.17, is different from the one defined at the functional level, and most of the *actions* may also be internally different. The estimation of the spacecraft position produces additional outputs and requires an additional input. The estimation algorithm is based on the numerical integration of the equations of motion, which accumulates error and so requires a periodical update of the orbit (position and velocity vectors). Also, magnetic torquers create magnetic dipoles and so their actuation commands must be expressed as dipoles and not torques. For this reason, algorithms that compute magnetic torquer actuation must receive as an input the measurement of the magnetic field vector in the body frame, to transform a desired torque into a command dipole (using Eq. D.27).

The specification of the *normal pointing* operation becomes more complete at the logical level (see Figure 4.18). With the components becoming less abstract, new *actions* are identified, revealing new interactions between the components (e.g., management of wheel saturation). In this mode, the attitude filter combines the measurement of angular velocity with those of the local magnetic field vector and the nadir. If reaction wheels are used, the attitude controller manages wheel saturation with torques applied by the magnetic torquers, which must therefore be counteracted by the reaction wheels. Both the actuation with and without reactions wheels is considered in the diagram as it was done at the functional level. The data report generation is identical to the one defined at the functional level.

The operation of the subsystem in the *safe pointing* mode, represented in Figure 4.19, differs from the operation in the *normal pointing* mode in two aspects. First, due to the high-power consumption of an Earth sensor, the estimation of the Sun vector replaces the measurement of the nadir. This means that the attitude filter must have a second configuration for this mode, in which the Sun vector and

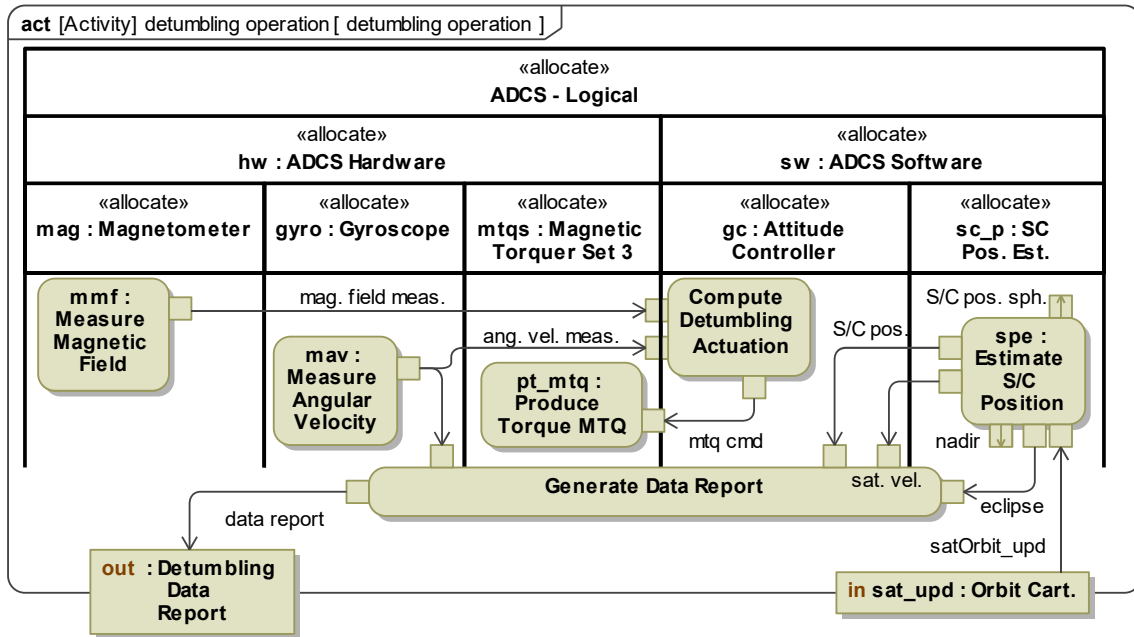


Figure 4.17: Interaction between logical components realizing the *detumbling operation* activity.

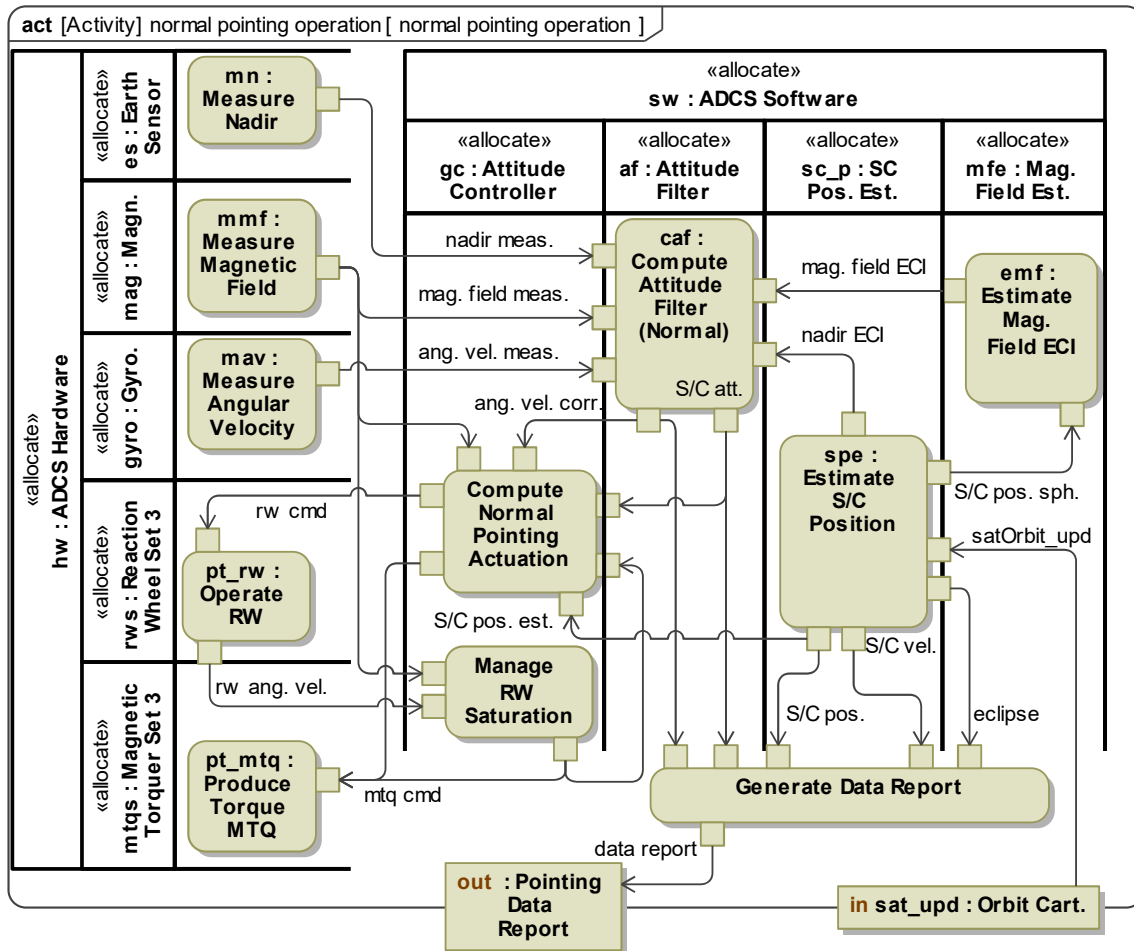


Figure 4.18: Interaction between logical components realizing the *normal pointing operation* activity.

the estimation of the Sun position in the ECI frame are used. The Sun position is estimated with a numerical integration of the Earth's orbit around the Sun, which also requires periodic updates. The second difference is that the actuation is only produced by the magnetic torquers (see Req. 7.1.1.1).

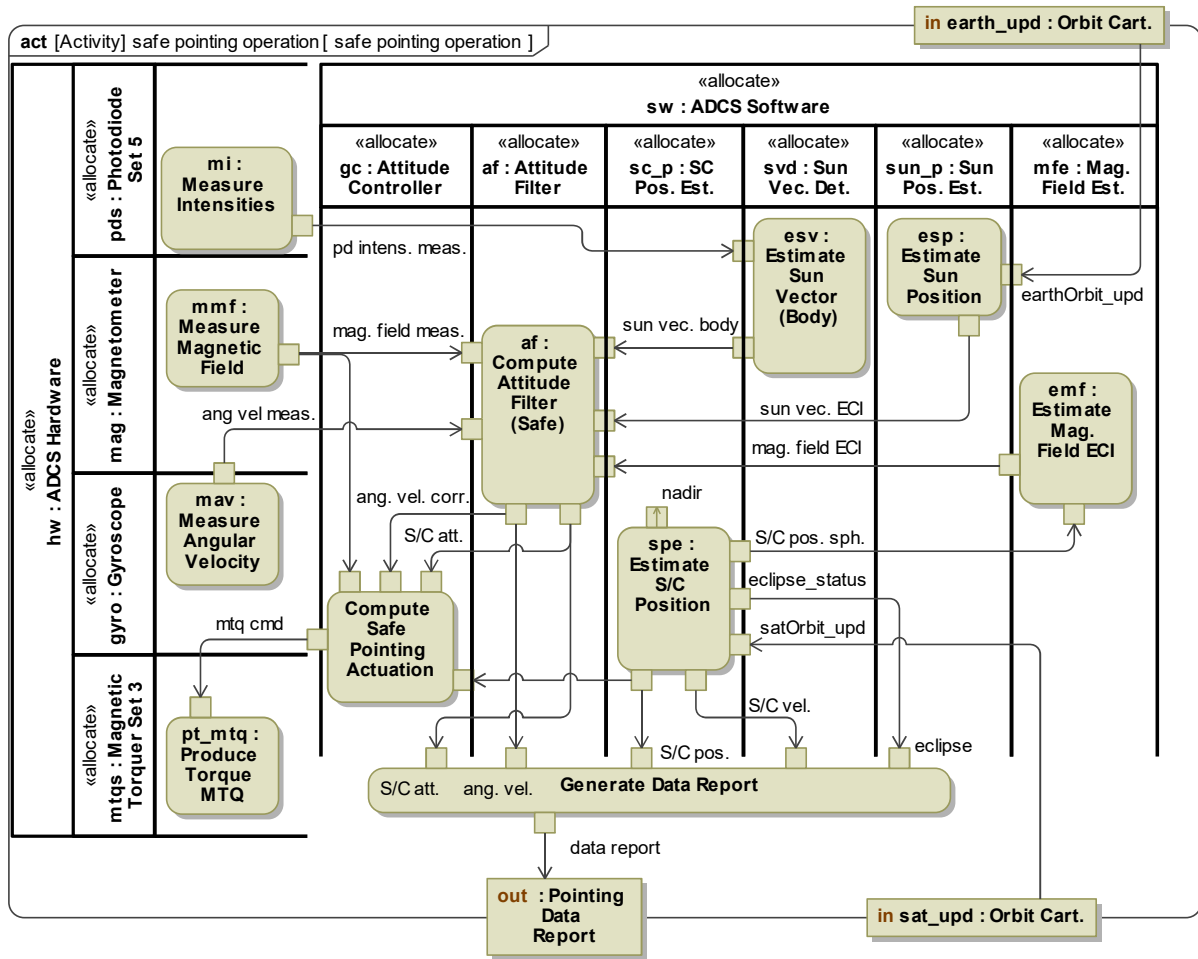


Figure 4.19: Interaction between logical components realizing the *safe pointing operation* activity.

As it was done the functional level, the internal structure of the logical architecture can be derived from the interactions shown above, while also considering the external interface of the subsystem. This internal structure is represented in an **ibd** diagram (see Figure 4.20). Each of the sensors interacts with the environment regarding the quantity that it measures (e.g., the *Magnetometer* block has an input representing the actual magnetic field vector in the body frame, and an output representing its measurement). Additionally, each sensor and actuator has an input *port* representing its electrical supply. As described in the process shown in Figure 3.4, software and hardware architectures are defined for the subsystem, which may then be integrated with those of the other subsystems.

In comparison with what was defined at the functional level, four *ports* were added to the ADCS. Two of these are inputs, which are related to the reception of spacecraft and Earth orbit updates, and the other two are outputs of the spacecraft position and attitude estimates. Since there are performance requirements imposing constraints on the errors of these two estimates, the two time-dependent parameters must be inputs of the *Monitoring and Control* block, and these connections must be routed through



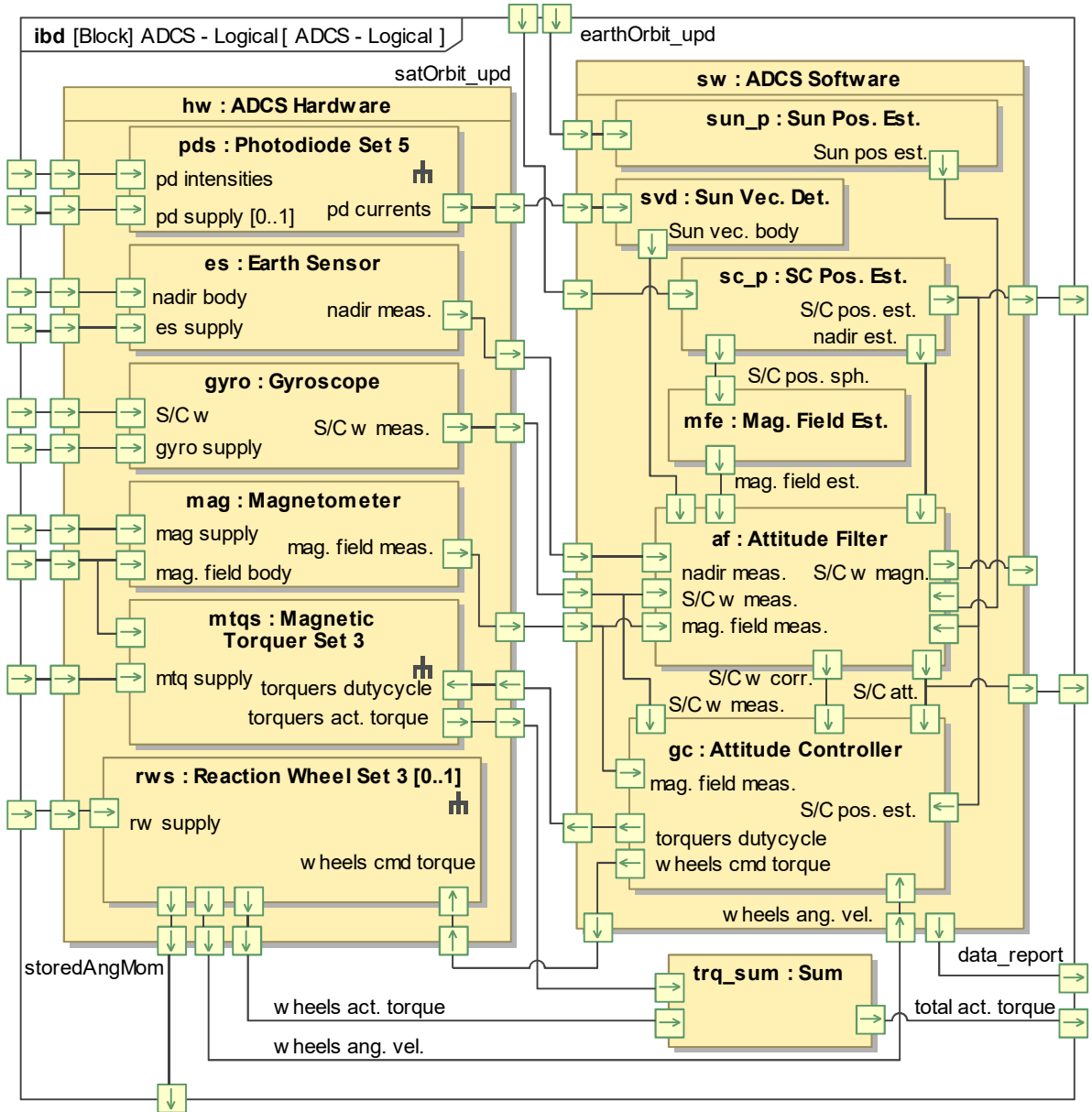


Figure 4.20: Internal structure of the ADCS logical architecture.

all the architecture levels (this is explained in Section 3.1.3 and shown in Figure 4.24).

Also as it was done at the functional level, each of the *token* exchanges specified in the diagrams in Figures 4.17 to 4.19 are assigned to a single connector represented in the **ibc** shown in Figure 4.20. This is done both as a good methodological practice and to enable proper integration with simulation. In fact, this **ibc** represents the exact structure of the ADCS in the simulation model, and the semantics used in its definition are very similar to those used in Simulink.

As the subsystem definition becomes less abstract, its external context may have to be redefined. In this case, this was required at the logical level. It represents a design iteration that is essential for the correct definition of the system, but which falls outside the linear and sequential representation of the system definition process. This type of iteration can occur, and it should take place whenever it contributes to the improvement of the design process effectiveness and efficiency. In this context, the

internal structure of the *Mission Context* block was redefined, as shown in Figure 4.21.

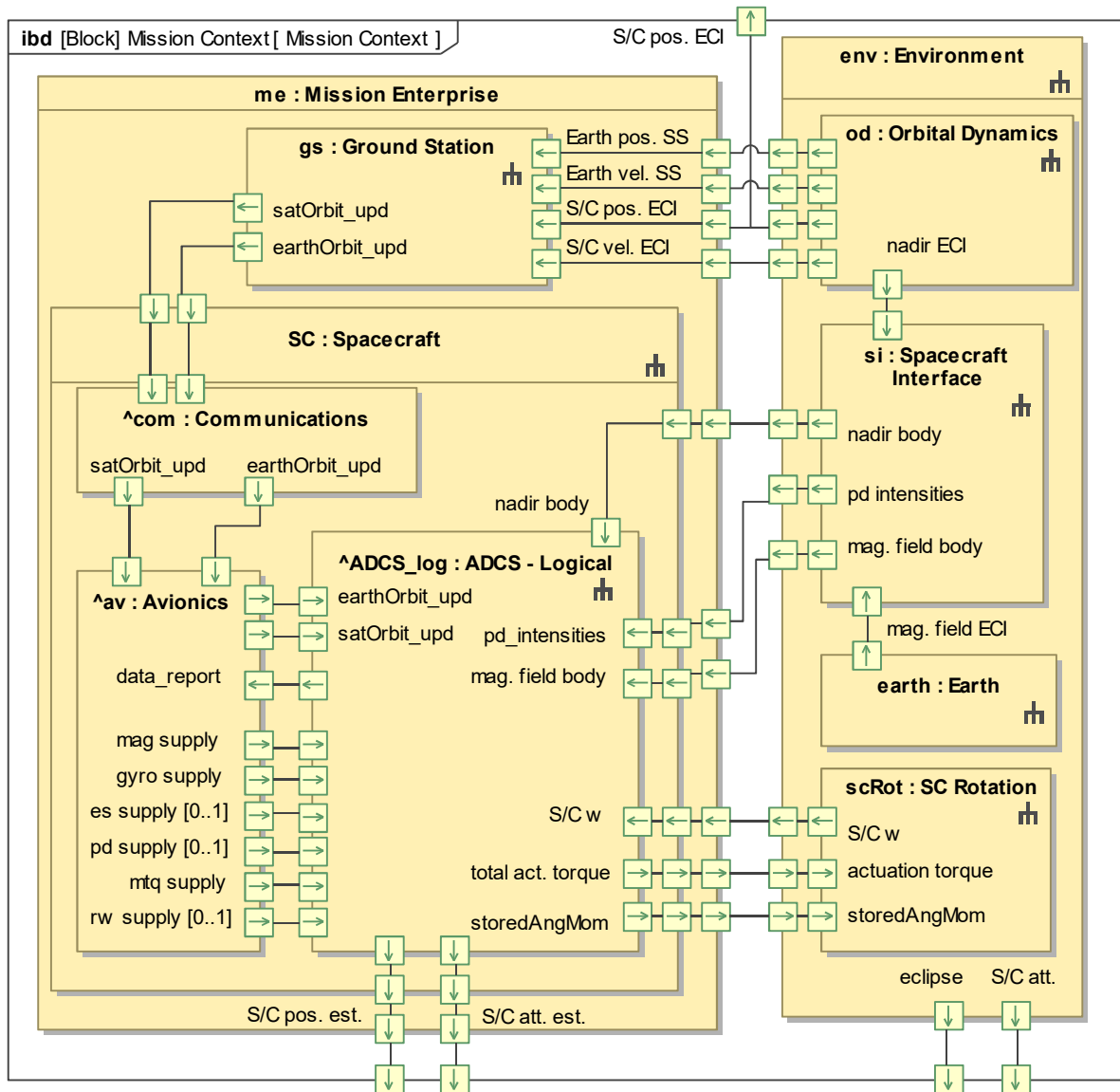


Figure 4.21: Redefinition of the external context of the ADCS.

The major change to the subsystem context regards the introduction of the spacecraft and Earth orbit updates. These represent a complex interaction which starts with the introduction of a new connection between the *Environment* and *Ground Station* blocks (the latter includes the ground station systems and the mission operations). Models of the orbits are used by an operator to generate orbit updates, which are then sent by the ground station to the spacecraft. The communications subsystem receives the updates and sends them to ADCS (with a relay in the avionics subsystem). The other change to the ADCS context concerns the breakdown of the electrical supplies. Output *ports* were also added to the *Mission Context* block, representing the time-dependent parameters that are required for ADCS performance evaluation, implemented through simulation (this becomes clear in Figure 4.24).

In Figure 4.21, the interactions between the system and the environment are also routed inside the *Environment* block. In this work, the environment is decomposed and modelled just like the subsystem.

This is done to ensure that the specification of the environment models is contained inside the SysML model repository. The internal structure of the *Environment* block is defined in detailed in Section 4.2.6.

The internal structure of the *Mission Context* block shown in Figure 4.21 is replicated in the simulation model that will be generated. This however does not mean that all the interactions modelled are relevant in simulation. In fact, the electrical supplies of the hardware components are irrelevant in the simulation process, but their presence in the model does not affect the results of the model transformation.

The capture of critical component properties, which are defined as *value properties* of the component blocks, is part of this activity (see Figure 3.4). However, this will only be presented in the next section, together with the definition of values for each attribute, facilitating reader understanding. Additionally, the logical component requirements, which are specified as part of this activity, will only be discussed in Section 4.3, since these are in practice design process outputs.

## 4.2.4 Synthesis of Candidate Physical Architectures

As discussed in Section 3.1.2, the activity of synthesizing candidate physical architectures consists solely of defining values for each attribute and redefining component multiplicities. Not all these attributes are quantitative since some of them represent the name of specific models or methods that are used as part of the definition of the component. This modelling practice was adopted to enable a real difference in the definition of component abstraction between the logical and physical levels, while also avoiding the additional work of redefining all the components between these two levels.

Not redefining the components is however just a simplification, which would not be possible in the context of a real project. For example, the quantitative attributes of an attitude filter cannot all be identified without an algorithm being selected. Nevertheless, this simplification contributes to an important reduction of scope, and it does not affect the validity of the methodology or MBSE in general, since there are notable methodologies that only define two component abstraction levels (like the original OOSEM).

Only two candidate architectures are defined at this level, exploring the two possible multiplicities of the set of reaction wheels. These two architectures consist of designs without and with one set of three reaction wheels. More candidates could also be defined by using different component options, but this is not as interesting for the validation of the methodology (as discussed in Section 3.1.2).

Three attributes of the subsystem are defined at this level. Concerning the angular rate thresholds described in Section 4.2.1, *PointThresh* is defined as 1 deg/s, and *DetThresh* is defined as 5 deg/s. The period of the state machine — *stmPeriod* — is defined as 10 seconds. These values were iterated in simulation. The duration of the detumbling, and the maximum stored angular momentum, also specified as *value properties* of the subsystem, will be later obtained with the help of simulation (in Section 4.2.6).

The critical properties of the hardware components are presented in Tables 4.3 and 4.4. Each column represents one component, for which a COTS component was selected, and whose name is shown in the first line of the table. For each *value property* a default value is defined, according to what is specified in the COTS component's datasheet. The attributes are divided into two table segments, the top one concerning properties that are relevant inside the SysML model, and the bottom one concerning prop-

erties used in the mathematical model of the component. These mathematical models are described in Appendix D.1, where the meaning of each parameter becomes clear. The direction of the actuation produced by each actuator is not defined in the tables, since it also not directly specified in the model. These are not properties of the COTS components, since they depend on the way these are installed in the spacecraft. They are however required for the corresponding mathematical models, and so these are connected to other *value properties* in the model, as explained in Section 3.1.3. Despite this, the actuators are considered to be aligned with each axis of the body frame, for simplicity. No information has been obtained regarding the measurement model of the photodiode, in specific regarding its accuracy. As a result, it was modelled as ideal in Simulink, since its real accuracy is not relevant for the validation of the methodology (a very large measurement error is already introduced by the Earth albedo).

| <b>Earth Sensor:</b><br><i>CubeSense</i> [87] |       | <b>Gyroscope:</b><br><i>MPU9250</i> [88]         |                    | <b>Magnetometer:</b><br><i>HMC5983</i> [89] |                   |
|-----------------------------------------------|-------|--------------------------------------------------|--------------------|---------------------------------------------|-------------------|
| Attribute                                     | Value | Attribute                                        | Value              | Attribute                                   | Value             |
| Mass (g)                                      | 30    | Mass (g)                                         | 1                  | Mass (g)                                    | 1                 |
| Avrg. Power (mW)                              | 100   | Avrg. Power (mW)                                 | 10                 | Avrg. Power (mW)                            | 0.5               |
| Supply Voltage (V)                            | 3.3   | Supply Voltage (V)                               | 2.5                | Supply Voltage (V)                          | 2.5               |
| Frequency (Hz)                                | 1     | Frequency (Hz)                                   | 40                 | Frequency (Hz)                              | 40                |
| FOV (deg)                                     | 130   | RRW Bias (deg/s)                                 | $6 \times 10^{-3}$ | Noise Variance (nT <sup>2</sup> )           | $4.0 \times 10^4$ |
| Accuracy 1- $\sigma$ (deg)                    | 0.2   | RRW PSD ((deg/s <sup>2</sup> ) <sup>2</sup> /Hz) | $2 \times 10^{-8}$ | Linearity                                   | 0.1%              |
|                                               |       | ARW PSD ((deg/s <sup>2</sup> ) <sup>2</sup> /Hz) | $1 \times 10^{-4}$ | Bias (nT)                                   | 100               |
|                                               |       | Linearity                                        | 0.1%               | No. Bits                                    | 12                |
|                                               |       | Saturation (rad/s)                               | 4.36               | Saturation (nT)                             | $1.0 \times 10^5$ |
|                                               |       | No. Samples MAF                                  | 10                 | No. Samples MAF                             | 10                |

Table 4.3: Critical properties of the Earth sensor, gyroscope, and magnetometer.

| <b>Reaction Wheel:</b><br><i>CubeWheel</i> [90] |                      | <b>Magnetic Torquer:</b><br><i>EnduroSat</i> [91] |       | <b>Photodiode:</b><br><i>EnduroSat</i> [91]    |                    |
|-------------------------------------------------|----------------------|---------------------------------------------------|-------|------------------------------------------------|--------------------|
| Attribute                                       | Value                | Attribute                                         | Value | Attribute                                      | Value              |
| Mass (g)                                        | 60                   | Mass (g)                                          | 9     |                                                |                    |
| Avrg. Power (mW)                                | 150                  | Avrg. Power (mW)                                  | 40    |                                                |                    |
| Max. Power (mW)                                 | 650                  | Max. Power (mW)                                   | 230   |                                                |                    |
| Supply Voltage (V)                              | 3.3                  | Supply Voltage (V)                                | 3.3   |                                                |                    |
| Wheel Inertia (kgm <sup>2</sup> )               | $2.1 \times 10^{-6}$ | Max. dipole (Am <sup>2</sup> )                    | 0.131 | Frequency (Hz)                                 | 1                  |
| Max. Ang. Rate (rad/s)                          | 837.76               | Dipole Direction                                  | —     | Conv. Factor (V <sup>-1</sup> m <sup>2</sup> ) | $4 \times 10^{-8}$ |
| Max. Torque (Nm)                                | $2.3 \times 10^{-4}$ |                                                   |       |                                                |                    |
| Actuation Direction                             | —                    |                                                   |       |                                                |                    |

Table 4.4: Critical properties of the reaction wheel, magnetic torquer, and photodiode.

The mass and average power consumption of each component must be specified, in order to perform roll-ups of these properties and check if the associated subsystem requirements are satisfied. The photodiodes are an exception to this, because these do not consume power and their inclusion in the solar panels is not optional (the inclusion of a magnetic torquer is optional). The supply voltage of each component and the maximum power of each actuator are also specified, since these are necessary for

the design of the avionics subsystem. The frequencies shown in Table 4.3 represent the values that were selected for this specific design, and not the maximum frequencies specified in the datasheets. Both the gyroscope and the magnetometer are implemented with moving average filters (MAF), which output values averaged over a number of samples, at the low frequency of the attitude filter.

At this point we can conclude that the total mass of the ADCS with the inclusion of reaction wheels is 239 grams, and 59 grams otherwise. Since the mass limit imposed by Req. 7.1.4 is 100 grams, the architecture with reaction wheels does not satisfy this requirement by a large difference. If the inclusion of reaction wheels is necessary to satisfy the performance requirements, either a smaller COTS option is selected, or a trade-off must be made (or both). This is however outside the scope of this work. Regarding the total average power consumption, it can be estimated by summing the average power of every component presented in Tables 4.3 and 4.4. This results in an average power of 680 mW for the configuration with reaction wheels, and 230 mW for the other configuration. The same exact scenario applies to the average power, but in this case, a better estimate of this parameter should first be computed in simulation (e.g., the power drawn by the reaction wheels depends heavily on their operating angular speed, which depends on the characteristics of the system and operational scenarios).

Since the software components are developed, and not procured, their specification is different from that of the hardware components. The critical properties of the software components are presented in Tables 4.5 and 4.6. The mathematical models of the software components are described in Appendix D.2, where the meaning of each parameter becomes clear.

The magnetic field estimation in the inertial frame is performed using the WMM 2020 (see Appendix C.2), while the numerical integration of the spacecraft and Earth orbits uses the fourth-order Runge-Kutta method (see Appendix D.2.1). The order of the magnetic field model and the orbit perturbations considered for each numerical integration were iterated in simulation to limit the estimation error to a desired value. The initial Julian date and frequency of these three estimators are not defined in the table for the reason discussed above concerning Table 4.4. They are instead respectively defined in Table 4.1 and in Table 4.6 (the frequency of the estimators is equal to the frequency of the attitude filter).

| Magnetic Field Estimator |          | Spacecraft Position Estimator |                 | Sun Position Estimator |                 |
|--------------------------|----------|-------------------------------|-----------------|------------------------|-----------------|
| Attribute                | Value    | Attribute                     | Value           | Attribute              | Value           |
| Initial Julian Date      | —        | Initial Julian Date           | —               | Initial Julian Date    | —               |
| Frequency (Hz)           | —        | Frequency (Hz)                | —               | Frequency (Hz)         | —               |
| Model                    | WMM 2020 | Integration Method            | RK4             | Integration Method     | RK4             |
| Model Order              | 10       | Orbit Perturbations           | $J_{8,8}$       | Orbit Perturbations    | None            |
|                          |          | Step Size (s)                 | 10              | Step Size (s)          | 50              |
|                          |          | Update Period (s)             | $1 \times 10^4$ | Update Period (s)      | $1 \times 10^5$ |

Table 4.5: Critical properties of the estimators of the magnetic field and spacecraft and Sun positions.

The blocks representing the attitude filter and attitude controller have multiple *actions* allocated to them, as shown in Figures 4.17 to 4.19. Also, some of these *actions* are executed in different subsystem modes, which means that these components must have different configurations for different modes. As a consequence, the mathematical models for these components, specified in Simulink, use the ADCS

mode that is active in each instant in their definition. This is why the state machine described in Section 4.2.1 must be included in the definition of the simulation model, as a Stateflow block. The access to a state machine's active state inside another mathematical model is described in Section 3.1.3.

The Sun vector determination is not shown in any table because the algorithm used is very simple and does not have any identifiable critical property (see the algorithm definition in Appendix D.2.1).

The attitude filter is executed in *normal pointing* and *safe pointing* only, and it uses the Explicit Complementary Filter (ECF) defined in Appendix D.2.1 (also used in the ISTSat-1). The values used for the gains in each configuration, whose meaning is described in Appendix D.2.1, are shown in Table 4.6 (these have been iterated in simulation). Also, different filter formulations are used in each mode since the accuracy of the Sun vector measurement is very poor. The formulation used in *normal pointing* is defined in Eq. D.16, and in *safe pointing* Eq. D.17 is used (the magnetometer is sensor *a*).

| Attitude Filter                                   |       | Attitude Controller                          |                    |
|---------------------------------------------------|-------|----------------------------------------------|--------------------|
| Attribute                                         | Value | Attribute                                    | Value              |
| Frequency (Hz)                                    | 1     | Frequency (Hz)                               | 1                  |
| Algorithm                                         | ECF   | Detumbling Gain                              | $2 \times 10^{-5}$ |
| Magnetometer Gain [ <i>normal</i> ]               | 0.3   | Proportional Gain [ <i>normal</i> ] (Nm)     | $2 \times 10^{-5}$ |
| Earth Sensor Gain [ <i>normal</i> ]               | 0.7   | Derivative Gain [ <i>normal</i> ] (Nm s/deg) | $1 \times 10^{-6}$ |
| Total Gain [ <i>normal</i> ] (rad/s)              | 1     | Off-loading Gain [ <i>normal</i> ] (Nm/rpm)  | $1 \times 10^{-8}$ |
| Bias Gain [ <i>normal</i> ] (rad/s <sup>2</sup> ) | 0.1   | Proportional Gain [ <i>safe</i> ] (Nm)       | $1 \times 10^{-6}$ |
| Magnetometer Gain [ <i>safe</i> ]                 | 1     | Derivative Gain [ <i>safe</i> ] (Nm s/deg)   | $1 \times 10^{-6}$ |
| Sun Vector Gain [ <i>safe</i> ]                   | 0.1   | Rotation Gain [ <i>safe</i> ] (Nm)           | $1 \times 10^{-6}$ |
| Total Gain [ <i>safe</i> ] (rad/s)                | 1     | Max. dutycycle                               | 0.75               |
| Bias Gain [ <i>safe</i> ] (rad/s <sup>2</sup> )   | 0.1   | Max. Torquer Dipole (Am <sup>2</sup> )       | —                  |
|                                                   |       | Max. Wheel Torque (Nm)                       | —                  |

Table 4.6: Critical properties of the attitude filter and attitude controller.

Regarding the attitude controller, its operation is different for each of the candidate architectures. Since the difference resides in whether reaction wheels are included, the attitude controller's operation is only different in what concerns the *normal pointing* mode. This difference is modelled by creating a *value property* owned by the *Attitude Controller* block of type boolean, which is *true* if reaction wheels are used or *false* otherwise. This parameter is used in the definition of the attitude controller mathematical model to enable the execution of the correct configuration, according to the SysML specification.

The attitude controller has three different configurations for the three different modes. In all configurations, the period of control, defined as 1 second, matches the frequency of the attitude filter. In *detumbling* mode, the simple algorithm defined in Eq. D.20 is used, where  $k_d$  is the detumbling gain shown in Table 4.6. In *safe pointing* mode, the pointing algorithm used in the ISTSat-1 is adopted, which is based on magnetic torquer actuation (formulated in Eq. D.24). The gains  $k_r$ ,  $k_e$ , and  $k_\omega$  correspond to the rotation, proportional, and derivative gains shown in the table, respectively.

In *normal pointing* mode, if reaction wheels are used, the basic PD control formulation defined in Eq. D.22 is used. In parallel, a magnetic torquer commanded dipole is calculated, being proportional to the angular momentum stored by the wheels (using Eq. D.28). The gain value used for this calculation is

also in Table 4.6. The torque produced by this dipole must be counteracted by an additional component of reaction wheel commanded torque. If reaction wheels are not used, the exact same controller formulation and gain values used in *safe pointing* mode are also used in this mode.

As mentioned in Section 2.6, the magnetometer measurements are affected by the magnetic torquer dipole, and because of that, these measurements and torquer actuation must be alternated. While the maximum duty cycle the torquers may produce is 1, a lower maximum is defined to represent that actuation is produced only in one part of the control period. In Table 4.6 this parameter is defined as 0.75, representing that 75% of the total period of 1 second is dedicated to magnetic torquer actuation.

All the controller gain values used in the different modes were iterated in simulation up-to the point that the performance requirements were met. Since these were not optimized in simulation, the comparison between the two different candidates is not completely accurate. But again, the purpose of this work is not to obtain an ideal ADCS design, but to apply the proposed methodology. In fact, the simulator that is generated in this work can be used in support of a gain optimization just like any other simulator.

#### 4.2.5 Management of Requirement Traceability

As discussed in Section 3.1.2, only part of the *Management of Requirement Traceability* activity can be meaningfully demonstrated in this work, namely the specification of which elements in the model satisfy and verify each requirement. These relationships were modelled and are summarized in Figure 4.22.

Requirements can be satisfied by a *value property* of an element (e.g., the requirements targeting the mass and average power consumption of the ADCS). A special case of this are the performance requirements, which are satisfied by *value properties* that represent TPMs. Each TPM is a scalar parameter that is constrained by a requirement, and whose definition should be clear in the description of the requirement, e.g., by defining confidence intervals and statistical interpretations of errors (this topic is further discussed in Appendix B). Other requirements are satisfied by *part properties*, specifically by the design description of the element in question. For example, the design constraint 7.4.2, which defines that the ADCS shall use active control techniques, is satisfied by the description of the component that represents the attitude controller, as shown in Figure 4.22.

Requirements may also be satisfied by behaviours attributed to elements, such as *operations* and classifier behaviours (or one of the state machines and *activities* that compose them). Some of the requirements are satisfied by the ADCS *operations*, while others are satisfied by the *activities* defined as *do* behaviours of the states (see Figure 4.22).

In this work, some requirements are verified through simulation. Performance requirements are directly verified through the determination of TPM values, following the process explained in Section 3.1.3, and other requirements are verified indirectly, by manual analysis of simulation results (e.g., the requirement concerning detumbling duration). The verification through simulation is modelled by assigning the *analysis context* block to the “verified by” property of the requirement. The requirements that are not verified through simulation may be verified with design reviews, for example. For these, the “verified by” property of the requirement is not specified in the model.




























































| Name                                                                                                                | Satisfied By                                                                                                                                                                                                                                                                                                                       | Verified By                                                                                                  |
|---------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
|  Safe Pointing                     |  safe pointing operation                                                                                                                                                                                                                          |                                                                                                              |
|  Mode Transition Triggering        |  forceModeTransition( newMode )<br> ADCS Main Behavior                                                                                                           |                                                                                                              |
|  Mode Mapping                      |  ADCS Main Behavior Complete                                                                                                                                                                                                                      |                                                                                                              |
|  Autonomous Mode Transition        |  enableAutonomousModeTransitions()<br> disableAutonomousModeTransitions()                                                                                        |                                                                                                              |
|  Transition to Pointing            |  detumbling operation                                                                                                                                                                                                                             |                                                                                                              |
|  Ancillary Data                    |  detumbling operation<br> normal pointing operation<br> safe pointing operation |                                                                                                              |
|  ADCS Mass                         |  mass_ADCS                                                                                                                                                                                                                                        |                                                                                                              |
|  Design Parameter Update           |  setParam( paramID, value )<br> getParam( paramID )                                                                                                              |                                                                                                              |
|  Periodic Parameter Updates        |  parameterUpdatePeriod                                                                                                                                                                                                                            |                                                                                                              |
|  Detumbling                        |  detumblingDuration                                                                                                                                                                                                                               |  ADCS_Pointing_Simulation |
|  Launcher Separation Rate          |  ADCS_phy : ADCS - Physical 1 = ADCS                                                                                                                                                                                                              |  ADCS_Pointing_Simulation |
|  ADCS Coverage                     |  ADCSCoverage                                                                                                                                                                                                                                     |  ADCS_Pointing_Simulation |
|  Wheel off-loading                 |  maxStoredAngMom                                                                                                                                                                                                                                  |  ADCS_Pointing_Simulation |
|  Average power consumption         |  avrgPower_ADCS                                                                                                                                                                                                                                   |                                                                                                              |
|  Absolute attitude normal pointing |  absPointingNormalPerf                                                                                                                                                                                                                            |  ADCS_Pointing_Simulation |
|  Absolute attitude safe pointing   |  absPointingSafePerf                                                                                                                                                                                                                              |  ADCS_Pointing_Simulation |
|  Absolute attitude knowledge       |  absAttitudeKnowledgePerf                                                                                                                                                                                                                         |  ADCS_Pointing_Simulation |
|  Orbit Knowledge                   |  orbitKnowledgePerf                                                                                                                                                                                                                               |  ADCS_Pointing_Simulation |
|  OBC Interaction                 |  Allocate[ADCS Software -> OBC]                                                                                                                                                                                                                 |                                                                                                              |
|  Control Scheme                  |  gc : Attitude Controller                                                                                                                                                                                                                       |                                                                                                              |
|  Attitude Determination          |  af : Attitude Filter                                                                                                                                                                                                                           |                                                                                                              |
|  ADCS Design Space               |  ADCS_phy : ADCS - Physical 1 = ADCS                                                                                                                                                                                                            |                                                                                                              |
|  Spacecraft Position Estimation  |  sc_p : SC Pos. Est.                                                                                                                                                                                                                            |                                                                                                              |

Figure 4.22: Specification of which elements in the model satisfy and verify each requirement.

## 4.2.6 Evaluation of Alternatives

This activity concerns system analysis, and in specific performance simulation, which is needed for requirement verification and comparison of physical architectures. The first step in the process used for this activity, shown in Figure 3.6, is to identify the analyses to be performed. In this work, one analysis is considered: the simulation of the ADCS performance. The scenario used in this simulation starts with launcher separation, after which the spacecraft is *init* mode and the ADCS in *detumbling* mode. The spacecraft will later transition to *safe* mode and then to *normal* mode. In each of these, the corresponding pointing performance is evaluated. With this scenario, all the ADCS modes are evaluated. More details on the simulation scenario are provided later in this section.

Before performing engineering analysis, the system, the environment, and the *analysis context* must be specified, following the six steps discussed in Section 3.1.3. Up to this point, the system has been specified and attribute values have been defined for the mission- and system-levels in Section 4.1, and for the subsystem- and component-levels in Section 4.2.4. The ADCS component mathematical models are described in Appendix D. The next step concerns the specification of the environment.



## Environment Model

The architecture of the environment is specified according to the specific simulation in question. For the simulation of ADCS performance, this is defined as shown in the **ibd** in Figure 4.23. *Value properties* are attributed to the components to represent model parameters and system or mission attributes that are used in their mathematical models.

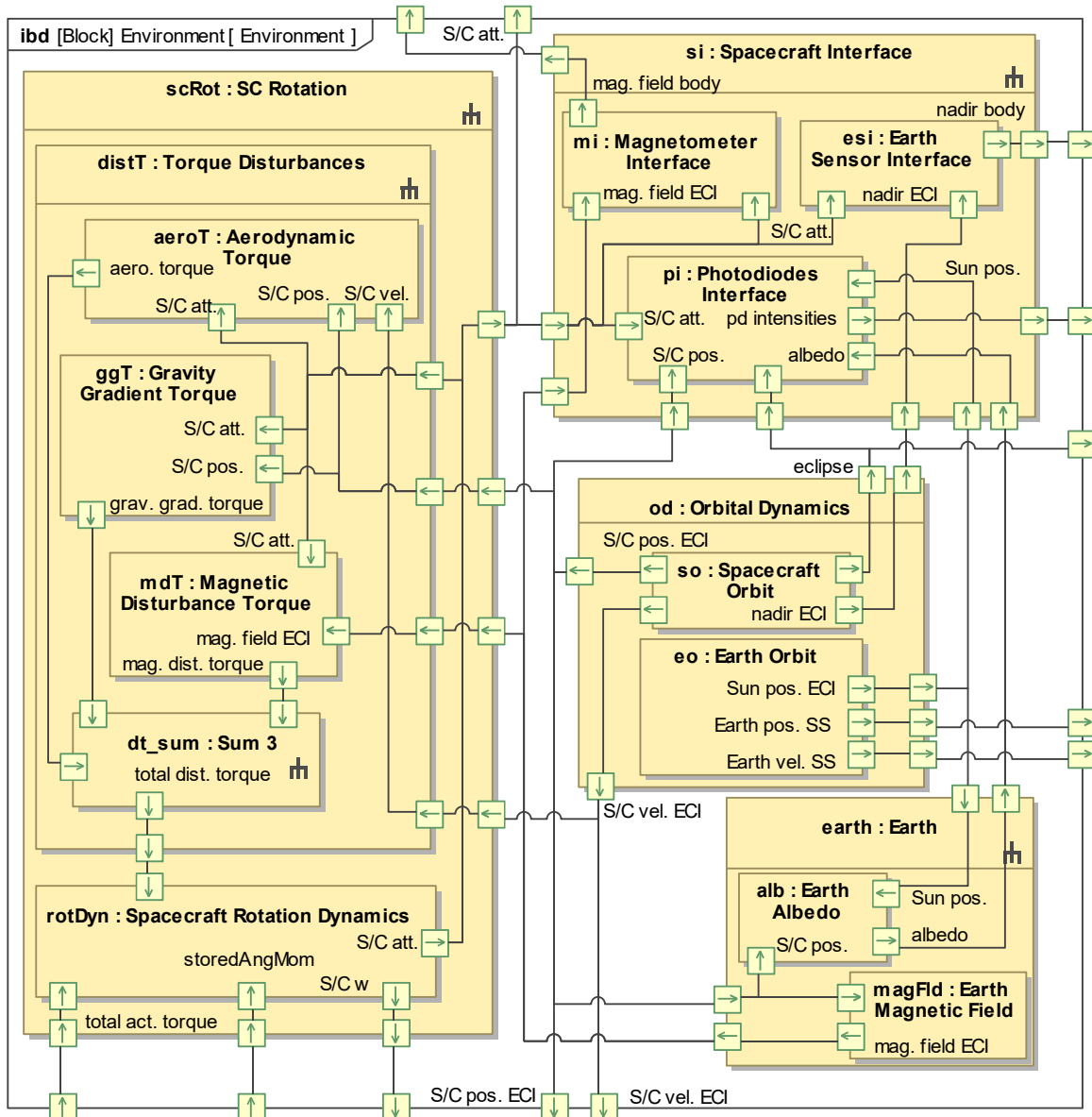


Figure 4.23: Internal structure of the *Environment* block.

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

Three torque disturbances are considered in simulation, namely the aerodynamic torque, gravity gradient torque, and magnetic disturbance torque (caused by undesired magnetic dipoles on-board).

These are the main sources of undesired torques in LEO [71], with slosh not being a problem since the spacecraft does not contain any fluids. The equations used to model the torque disturbances and rotation dynamics are presented in Appendix A.3. All the components related to the spacecraft rotation are executed in simulation with the smallest step-size, except for the computation of the aerodynamic torque, which is performed with a sampling period of 1 second.

The Earth magnetic field is modelled with the WMM 2020, using Gauss coefficients of up to degree and order 12 (highest available in [92]). The detailed formulation of this model can be consulted in Appendix C.2. In turn, the computation of the Earth albedo is based on the work developed in [93], and it is separated into two parts, as explained in Appendix C.4. The first part of the albedo computation is represented by the block *Earth Albedo*, and it is executed with a sampling period of 10 seconds. The second part of the computation, which requires the knowledge of the spacecraft attitude, is represented by the block *Photodiodes Interface*, and it is executed at the frequency of the attitude filter. This block outputs the exact quantity that is measured by the photodiodes, i.e., total irradiance. The two other blocks contained in the *Spacecraft Interface* only apply a coordinate transformation to the magnetic field vector and the nadir, since the sensors measure these quantities in the body frame.

Instead of developing a program for the precise determination of the spacecraft orbit, we opted to use the GMAT [94] software to this effect (version R2016a). This software was also used to compute the intervals of eclipse. All these computations are encompassed in the *Spacecraft Orbit* block. Regarding the propagation of the spacecraft orbit, the configurations used are listed in Table 4.7. To evaluate the order of magnitude of the various orbital perturbations, Figure 3.1 in [95] was used as a reference. All the perturbations that represent accelerations greater than  $10^{-10}$  km/s were considered, except for the dynamic solid tide, which is not modelled in the used version of GMAT. The Solar Radiation Pressure (SRP) perturbation was also considered because its effect increases with the decrease in spacecraft size, and the spacecrafts considered in [95] are much larger than a CubeSat.

|                                  |                                           |
|----------------------------------|-------------------------------------------|
| Orbit Perturbations Considered   | Moon, Sun, SRP, Drag, $J_{18,18}$ (up to) |
| Model of the Earth Gravity Field | EGM-96                                    |
| Atmosphere Model                 | MSISE90                                   |
| Integrator Type                  | Runge-Kutta 8/9                           |
| Step Size                        | 5 seconds                                 |

Table 4.7: Configurations used in GMAT for the determination of the spacecraft orbit.

Some spacecraft properties are required to specify the GMAT model: mass, drag coefficient, reflectivity coefficient, drag area, and SRP area. The values used for these parameters are those presented in Table 4.2. The entry orbit and associated epoch used in this model are defined in Table 4.1.

Finally, the orbit of the Earth around the Sun is propagated with a numerical integration of the equations of motion, considering only the gravity of the Sun and Moon (as point masses). The orbit of the Moon is also propagated in this process. The integration method used is a fourth-order Runge-Kutta (see formulation in Appendix D.2.1), and the step size used is 50 seconds. The initial position and velocity were generated using the MATLAB function *planetEphemeris()*, which uses the Chebyshev coefficients that the NASA JPL provides (in specific the DE405 ephemerides [96]).

## Analysis Context

After both the system and environment are modelled in SysML, the *analysis context* and the *Monitoring and Control* block are defined to specify the simulator, as discussed in Section 3.1.3. As shown in Figure 4.24, the *analysis context* is composed of the mission context and a *Monitoring and Control* block, like the simulator structure shown in Figure 2.5.

The *Monitoring and Control* block contains elements that compute specific error indices (defined in Appendix B), which are targeted by each of the performance requirements. These indices, which are later processed to generate TPM values, are: (i) the spacecraft position estimation error, computed as the magnitude of the difference between real and estimated positions; (ii) the absolute attitude knowledge error, computed with Eq. B.3; and (iii) the absolute pointing error, computed with Eq. B.4. Two different absolute pointing errors are calculated, one concerning the error in *safe pointing* mode and out of the eclipse, and the other concerning the error in *normal pointing* mode.

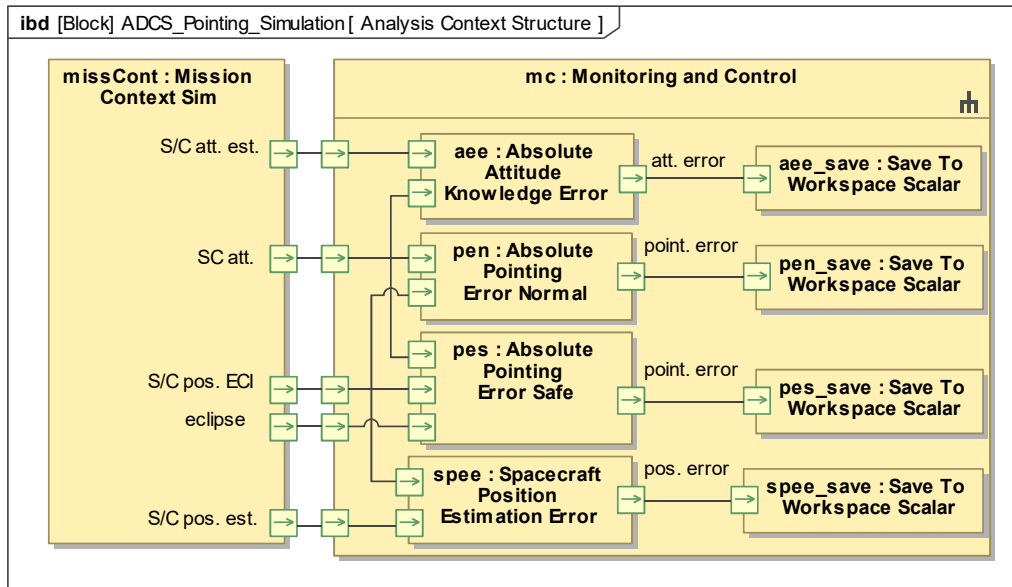


Figure 4.24: Internal structure of the *analysis context* and the *Monitoring and Control* block.

In simulation, all error indices are computed at every instant and saved (with a frequency of 1 Hertz). The four blocks that compute these indices all have a *value property* that specifies the interval of time in which the index values are relevant for the calculation of the final TPM.

As shown in Figure 4.24, the *Monitoring and Control* block is also composed of blocks that save the index values to the workspace in MATLAB, in structures fields whose names are specified in SysML. These save index values are later post-processed by functions that are called after the simulation, using with the *StopFCN* property of the Simulink model, as discussed in Section 3.1.3. Each function transforms a set of instantaneous values into one single scalar parameter, the TPM, which is computed using the definition in Eq. B.5. Only instantaneous values inside the specified interval of time are considered for this calculation. The final TPM value is saved in an Excel spreadsheet that is synchronised with a table in SysML, being automatically transferred to the SysML model.

As mentioned in Section 3.1.3 and defined in [60], the specification of the *Monitoring and Control*

block includes the definition of the simulation scenarios. These are modelled in SysML as a schedule of *signal events*, which are mapped to Simulink as messages. The messages are received by Stateflow blocks, guiding the state transitions according to the system operations scenario defined by the user.

One single simulation is run for each candidate architecture, starting with launcher separation and finishing with normal mission pointing. This simulation must verify all the requirements mentioned in Section 4.2.5, and to do this, the events shown in Table 4.8 must take place in the defined order. After separation, which takes place in the instant  $t = 0$  sec, the *init* mode is active for 90 minutes, after which a transition to *safe* mode occurs. The limit duration of the detumbling (i.e., ensuring the angular rate is below 5 deg/s) is however 2 complete orbital periods (the orbital period is 5668 seconds). As the detumbling time limit is reached, the ADCS has 2 complete orbital periods to converge the spacecraft stabilization, after which the pointing performance in *safe pointing* mode is evaluated. According to Req. 7.2.5, this evaluation must be performed during 10 complete orbits. After this is finished, a transition to *normal* state is triggered by ground command, and the same process is applied but now in *normal pointing* mode (2 orbital periods for convergence and 10 orbital periods for performance evaluation).

| Event                                               | Instant<br>(seconds) | Rationale                              |
|-----------------------------------------------------|----------------------|----------------------------------------|
| Launcher Separation                                 | 0                    | —                                      |
| Transition to SAFE                                  | 5400                 | 90 minutes after separation            |
| Detumbling Time Limit                               | 11336                | 2 orbits after separation (Req. 7.2.3) |
| <i>Safe Pointing</i> : Start of Performance Eval.   | 22672                | 2 orbits for stabilization             |
| <i>Safe Pointing</i> : End of Performance Eval.     | 79352                | 10 orbits of coverage (Req. 7.2.5)     |
| Transition to NORMAL                                | 80000                | —                                      |
| <i>Normal Pointing</i> : Start of Performance Eval. | 91336                | 2 orbits for stabilization             |
| <i>Normal Pointing</i> : End of Performance Eval.   | 148016               | 10 orbits of coverage (Req. 7.2.5)     |
| End of Simulation Scenario                          | 150000               | —                                      |

Table 4.8: Schedule of events representing the simulation scenario defined.

For this schedule of events to be put into practice, three *signal events* must be modelled in SysML, as shown in Figure 4.25. The signals *Separation\_Start* and *Separation\_Complete* are required to arrive at the spacecraft state *on* (see Figure 4.4). In turn, the signal *Ground\_Cmd\_Normal* defines the ground command that triggers a transition to *normal* in the exact instant specified in the table ( $t = 80000$  sec).

## Simulation Results

With system, environment, and *analysis context* completely specified, the simulator can be generated, following the process described in Section 3.1.3, and executed. One simulation was run for each candidate architecture, and the results of these are summarized in Table 4.9 (the step size used was 25 ms).

The detumbling duration and absolute pointing performance in *safe pointing* are equal for the two architectures, because they do not concern the *normal pointing* mode. The two requirements addressing these attributes are both satisfied by a large margin (see Table 4.9). The spacecraft position estimation

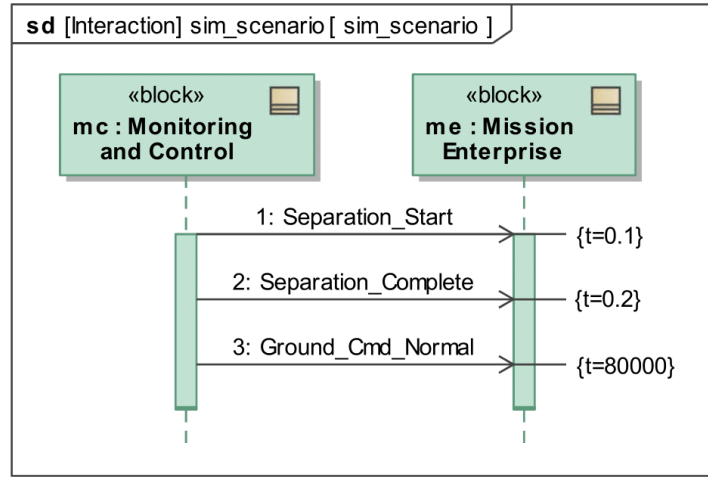


Figure 4.25: Specification of the simulation scenario with a schedule of *signal events*.

| Parameter                                                                    | Value | Required Max. |
|------------------------------------------------------------------------------|-------|---------------|
| [A1]: Absolute Pointing Performance – <i>normal pointing</i> (deg)           | 0.50  | 1             |
| [A2]: Absolute Pointing Performance – <i>normal pointing</i> (deg)           | 0.96  | 1             |
| [A1]: Absolute Attitude Knowledge Performance – <i>normal pointing</i> (deg) | 1.81  | 2.5           |
| [A2]: Absolute Attitude Knowledge Performance – <i>normal pointing</i> (deg) | 2.74  | 2.5           |
| Absolute Pointing Performance – <i>safe pointing</i> (deg)                   | 6.63  | 10            |
| Spacecraft Position Estimation Accuracy (m)                                  | 208   | 250           |
| Detumbling Duration (sec)                                                    | 5680  | 11336         |
| Maximum Wheel Angular Rate (rpm)                                             | 259   | 6000          |

Table 4.9: Summary of the simulation results (the architecture with reaction wheels is named “A1”, and the other “A2”). The parameter values obtained with simulation (left) are compared to maximum values imposed by requirements (right).

accuracy is also equal for the two architectures since changes in attitude determination and control do not affect this estimation. The associated requirement is also satisfied for both architectures (note that the performance of the spacecraft position estimation is evaluated during the entire simulation).

The absolute pointing performance in *normal pointing* is different in the two cases because different actuators are used. Even though the use of reaction wheels ensures a much lower pointing error, both architectures meet the imposed requirement (see Table 4.9). The absolute attitude knowledge performance in *normal pointing* is also different for the two architectures, not because of the attitude filters, which are exactly equal, but because the absolute pointing error has an effect on the attitude estimation accuracy. The corresponding requirement is only satisfied by the architecture with reaction wheels, but this requirement may be redefined if a trade-off with another requirement meets the stakeholder expectations better (e.g., with the mass requirement). When reaction wheels are used, the wheel angular rate is kept at very low levels, when compared with the maximum angular speed required (the requirement concerns the stored angular momentum, but this is proportional to the angular speed).

The TPM values shown in Table 4.9, in the first six lines, are the only relevant parameters for determining if the associated requirements are verified. However, simulation results should be analysed in detail by the specialist engineers to understand if the design of the subsystem may be improved. To

enable this, some time-dependent parameters are saved after the simulation to be analysed later.

In Figure 4.26 the absolute attitude knowledge error is plotted against time, for both simulations. 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. The nadir is always close to aligned with the z axis of the body frame, and the estimation error accumulates around this direction, which means these error spikes have a very low influence on the absolute pointing error.

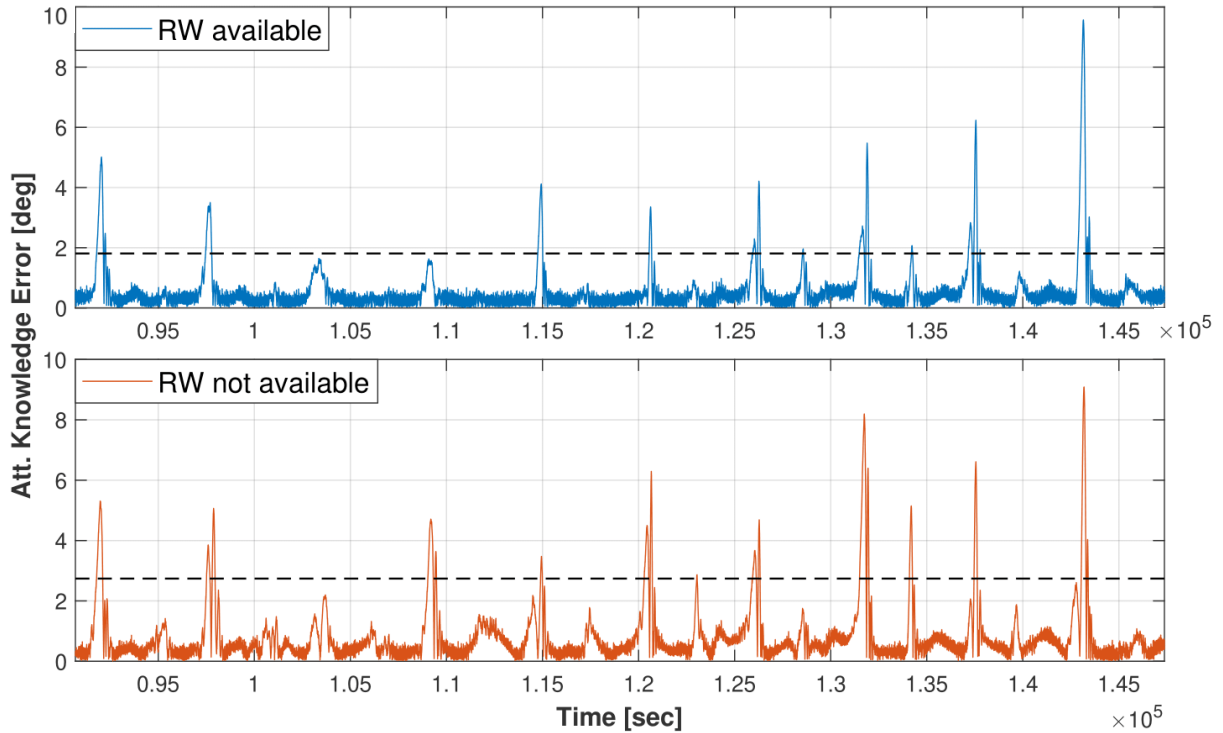


Figure 4.26: Absolute attitude knowledge error during the 10 orbits of performance evaluation for the *normal pointing* mode. The dashed line is located at the final value of the corresponding TPM.

This very low influence is demonstrated on the plots of absolute pointing error against time in *normal pointing* mode, shown in Figure 4.27. The architecture that includes reaction wheels 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 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.

Finally, in Figure 4.28 we show a plot of the absolute pointing error against time in *safe pointing* mode. The orange line represents the pointing error out of the eclipse and the blue line in eclipse. The pointing error is much higher in eclipse because only the magnetic field vector measurement is used by the attitude filter to correct the integration of the gyroscope measurements, which results in an accumulation of estimation error around the direction of the magnetic field vector measurement. When comparing the results shown in this figure with the ones shown in Figure 4.27, we may conclude that the difference in the magnitude of the pointing error is a consequence of the less precise attitude estimation, which is a result of the use of a Sun vector measurement in the place of the nadir measurement.

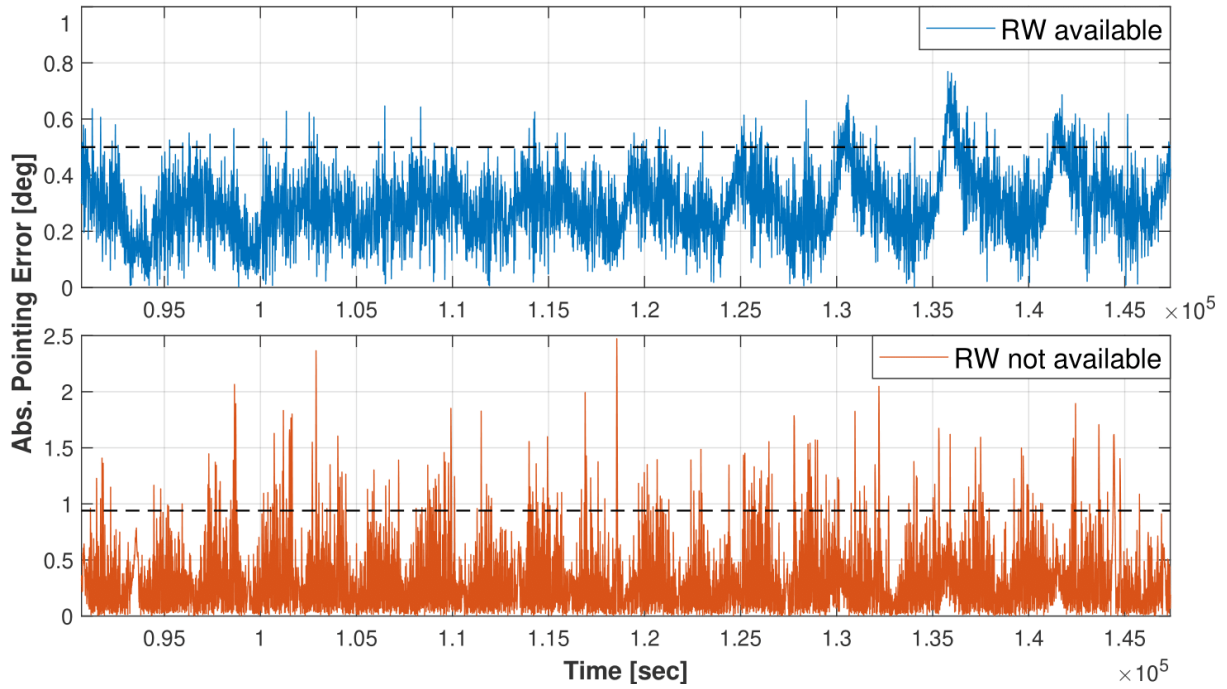


Figure 4.27: Absolute pointing error during the 10 orbits of performance evaluation for the *normal pointing* mode. The dashed line is located at the final value of the corresponding TPM.

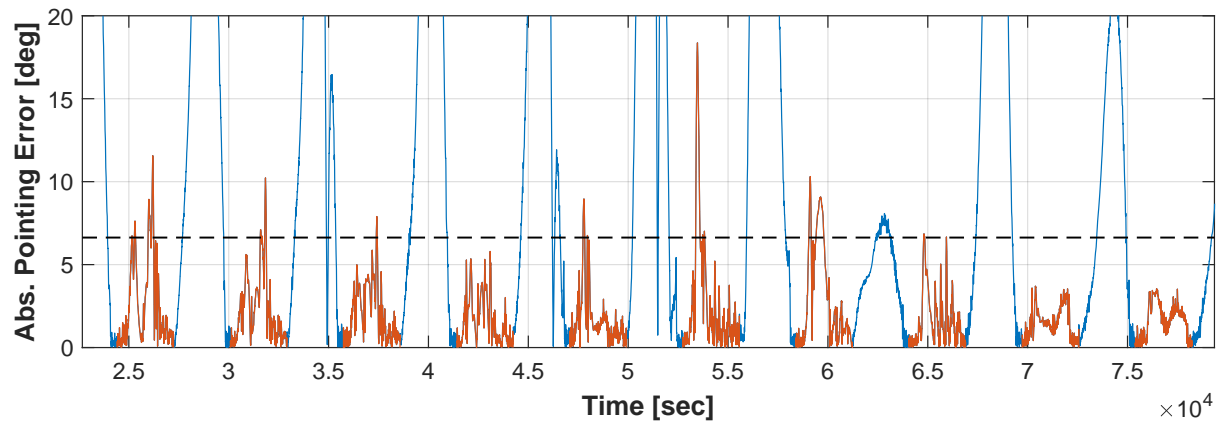


Figure 4.28: Absolute pointing error during the 10 orbits of performance evaluation for the *safe pointing* mode. The dashed line is located at the final value of the corresponding TPM.

### 4.3 Design Process Outputs

Most of the outputs of the design process specified in Figure 2.8 have already been defined in Section 4.2, specifically: (i) the subsystem white box specification, specified with the subsystem decomposition shown in Figure 4.16 and with the **ibd** shown in Figure 4.20; (ii) the subsystem behaviour specification, specified with the state machine shown in Figures 4.8 to 4.10 and with the *activities* shown in Figures 4.17 to 4.19; and (iii) the component black box specification, which can be directly derived from the diagram in Figure 4.20, which shows the external interface of each component, and from Tables 4.3 to 4.6, which specify the quantitative attributes of each component. The final output of this design process concerns the component requirements, whose derivation we discuss in this section.

Some component-level requirements are directly derived from subsystem-level requirements. The performance requirement that defines the accuracy of the orbit estimation (Req. 7.3.4) results directly into a component requirement, since only one component is involved in this estimation, the spacecraft position estimator. This component also requires periodic orbit updates, as does the sun position estimator. A requirement for each of these components is directly derived from Req. 7.2.2, imposing that the period of the orbit updates be greater than the maximum period without ground station contact.

Other component-level requirements are indirectly derived from subsystem-level requirements. As an example, we may consider that for the detumbling to be completed before the defined time limit given the worst-case launcher separation rate of 100 deg/s, there is a maximum frequency for the attitude controllers which can be defined with simulation. Thus, a requirement concerning the frequency of the attitude controllers may be derived from the requirements concerning the detumbling operation (Reqs. 7.2.3 and 7.2.4). This requirement is only relevant when considering that the frequency of the attitude controllers may have to be redefined in a later phase due to implementation constraints.

Additionally, budgets of certain parameters can be defined at the component-level as it was done at the subsystem-level. If the hardware components were developed in-house or if the selection of COTS options was not complete, the mass and average power budgets could be further broken down at the component-level. Other budgets may be defined concerning the software components, namely of processing time or memory. As part of the design of the ADCS, a control period of 1 second was defined, and this time must be enough to compute the spacecraft position, the sun position, the magnetic field vector, the spacecraft attitude, and the controller outputs. Since some of these computations are more complex than others, different maximum processing times may be defined for each.

Finally, requirements may also be identified during the design definition process. As an example, we may consider the fact that the magnetometer should not draw measurements while the magnetic torquers produce a magnetic dipole, since these will be affected by the dipole. As mentioned in Section 4.2.4, the maximum duty cycle represents the percentage of the control cycle dedicated for torquer actuation. From this interaction, it is identified that the attitude controller should ensure that the magnetic torquers only produce a dipole during the part of the control period assigned for that purpose. Additionally, the magnetometer frequency and the number of samples over which is applied an average must be constrained to ensure that measurements taken during the torquer actuation are not considered.

During the design definition process, other more detailed requirements may also be defined for the components, e.g., concerning the specification of the units for each sensor measurement, regarding the capability of the attitude filter to detect sensor failures, or imposing the configurability of specific parameters used in the software components algorithms.

## 4.4 Discussion of Results

Some of the benefits of MBSE mentioned in Section 1.2.3 were perceived through the development of this work. The improved ability to manage complexity is observed in the breakdown of the subsystem behaviour. The ADCS state machine is represented with three different automatically consistent **stm** dia-



grams (Figures 4.8 to 4.10), and the subsystem behaviour in each of its operating modes is represented in separate **act** diagrams (Figures 4.17 to 4.19). While DBSE methodologies may ensure the capability to produce this type of breakdown, they usually do not ensure automatic maintenance of consistency.

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. For example, the ports owned by every element contained in the *analysis context* sum up to more than 300. All of these are typed by an *interface block* that specifies the type of quantity that flows through it and the units of that quantity. The ability of MBSE to ensure automated correctness verification guarantees that all the pairs of ports that are connected in the model share the same definition in terms of quantity and units. In addition, if the user wants all the ports representing an angular quantity to change their units from radians to degrees, for example, this can be achieved by changing the specification of one *interface block*, and this singular change is propagated across the complete model.

MBSE also enables the reuse of existing models, something that is made clear by the reuse of the model developed in [21], which served as a basis for the model developed in this work. The model organization was adopted almost completely, and many other elements were reused, namely *stereotypes*, *value types* and signal definitions. With the reuse of this model, the total modelling effort was reduced, and the arduous task of setting up the model from scratch was avoided.

Regarding the methodology modifications proposed in this work, the introduction of a third level of component abstraction offers clear benefits. At the physical layer, the main focus of the design is on the use of simulation to compare different options of components and different algorithm configurations, with the purpose of selecting a final subsystem design. Before reaching this layer, it is important to separate into two different steps the definition of functionality and the selection of specific technology implementations. This way, we can ensure that: (i) systems engineers completely identify and decompose the subsystems functions, which helps ensuring minimum design scope and avoid rework; (ii) early design decisions that inadvertently remove a part of the design space are avoided; and (iii) the integration issues that emerge from the decrease in the abstraction level do not have to be resolved all at once, since there are instead two instances of abstraction reduction (between consecutive layers).

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. When changes were made to the specification in SysML, these were always promptly propagated to the simulation model. Also, all the mathematical models used in the simulation were completely developed in Simulink, a tool designed for this purpose. It was also demonstrated how an external simulation tool, the GMAT, can be integrated into the simulation model indirectly via MATLAB/Simulink.

Model transformation approaches are currently capable of mapping the structural decomposition and connections between elements, specified in SysML, into simulation platform-specific semantics. This part of the developed framework is quite mature, mostly because the semantics used to define these type of specification are very similar between the SysML and Simulink. 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 identified and 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. SysML *activities* are used, for example, to model algorithms and *interactions* between components that realize functions of the subsystem. Their semantics are quite different from Simulink semantics and, because of that, a mapping between the two is still not available. Because of this, the behaviour of the components is specified in SysML and simultaneously modelled mathematically in Simulink. Consequently, the component specification is not truly contained in a single source of truth since consistency between SysML and Simulink representations is ensured manually. However, if the existence of multiple sources of truth can be limited to the component-level of the architecture, model transformation approaches may still be advantageous. This is because, at this level, only the behaviour and black box representations of the components in SysML and Simulink must remain consistent, and because these components are usually reused from previous projects, in which they were already modelled consistently.

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* (see Section 2.2), 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. This mapping is already defined by the SysPhS standard, and it can be robustly implemented, because the SysML state machine and Stateflow semantics are also very similar. Currently, SysML service-based behaviour specification is not as well mapped to Simulink as state-based behaviour. However, in this work, we showed that message exchanges specified in SysML can also be transformed into Stateflow semantics (the *signal events* in Figure 4.25 are mapped to Stateflow messages in the final simulator).

The model transformation framework was also capable of generating a simulator that complies with the definition shown in Figure 2.5. Specifically, it ensures that the *Monitoring and Control* block can be fully specified in SysML. In fact, all the relevant TPM specification is contained in a single source of truth, including confidence intervals, statistical interpretation, and evaluation intervals. Also, the simulation scenario used for requirement verification was completely specified with *signal events* in SysML.

Model transformation approaches must become more mature for application in the development of space systems in an industry setting. This maturity can be increased with a continuous improvement of the model transformation standards, such as the SysPhS, and by modifying the SysML standard, making it more suitable for the specification of execution semantics.

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.

## Chapter 5

# Conclusions

Model-based systems engineering has the potential to further improve the effectiveness of SE practices, by introducing modelling and centralization of information into these. However, a few challenges must be first overcome, before MBSE may fulfil its promise. Some of these challenges are the lack of practical guidance on the application of MBSE, the lack of maturity of tools and methods, and the limitations on tool integration, specifically concerning 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. This type of approach consists of the automatic transformation of system models contained inside the model repository into simulation models that can be executed in a simulation environment. The developed framework is capable of generating a complete simulator from SysML specification, reusing component mathematical models developed in Simulink. 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. A preliminary iteration of the develop model transformation framework was proposed and demonstrated in a paper presented in the 2021 Simulation and EGSE for Space Programmes (SESP) workshop, organized by the ESA Conference Bureau.

Some of the benefits of MBSE identified across literature were verified in the implementation of the methodology, namely: (i) the improved ability to manage complexity; (ii) the improvement in quality obtained with automatic propagation of changes and model correctness evaluation; and (iii) the reduction in the SE effort introduced by the ability to reuse models. Also, 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 re-configured 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.

# Bibliography

- [1] Tamaskar, S., Neema, K., and DeLaurentis, D., “Framework for Measuring Complexity of Aerospace Systems,” *Research in Engineering Design*, Vol. 25, No. 2, 2014, pp. 125–137. doi: 10.1007/s00163-014-0169-5.
- [2] Flood, R. L., and Carson, E., *Dealing with Complexity: An Introduction to the Theory and Application of Systems Science*, Springer US, New York, NY, USA, 1993. ISBN: 978-1-4419-3227-3.
- [3] Walden, D. D., Roedler, G. J., Forsberg, K. J., Hamelin, R. D., and Shortell, T. M. (eds.), *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 4<sup>th</sup> ed., John Wiley & Sons Inc., Hoboken, NJ, USA, 2015. ISBN: 978-1-11-899940-0.
- [4] Larson, W. J., Kirkpatrick, D., Sellers, J. J., Thomas, L. D., and Verma, D. (eds.), *Applied Space Systems Engineering*, McGraw-Hill Companies Inc., Boston, MA, USA, 2009. ISBN: 978-0-07-340886-6.
- [5] Alvarez, J. L., de Koning, H., Fischer, D., Wallum, M., Metselaar, H., and Kretzenbacher, M., “Best Practices for Model Based Systems Engineering in ESA Projects,” *2018 AIAA SPACE and Astronautics Forum and Exposition* (17-19 September 2018, Orlando, FL, USA), 2018. doi:10.2514/6.2018-5327.
- [6] ECSS Secretariat, “Space Engineering - System Engineering General Requirements,” Tech. Rep. ECSS-E-ST-10C, ESA-ESTEC Requirements and Standards Division, Noordwijk, The Netherlands, 2017.
- [7] ISO/IEC/IEEE, “Systems and Software Engineering — System Life Cycle Processes,” Tech. Rep. 15288:2015, International Organization for Standardization, Geneva, Switzerland, 2015.
- [8] Elm, J. P., and Goldenson, D. R., “Quantifying the Effectiveness of Systems Engineering,” *2013 IEEE International Systems Conference (SysCon)* (15-18 April 2013, Orlando, FL, USA), 2013, pp. 6–13. doi:10.1109/SysCon.2013.6549850.
- [9] Boehm, B., Valerdi, R., and Honour, E., “The ROI of Systems Engineering: Some Quantitative Results for Software-Intensive Systems,” *Systems Engineering*, Vol. 11, No. 3, 2008, pp. 221–234. doi:10.1002/sys.20096.

- [10] Kasser, J., and Hitchins, D., “Yes Systems Engineering, You Are a Discipline,” *INCOSE International Symposium*, Vol. 22, No. 1, 2012, pp. 416–431. doi:<https://doi.org/10.1002/j.2334-5837.2012.tb01346.x>.
- [11] Hitchins, D. K., *Systems Engineering: A 21st Century Systems Methodology*, John Wiley and Sons Ltd., Chichester, UK, 2007. ISBN: 978-0-470-05856-5.
- [12] Dick, J., Hull, E., and Jackson, K., *Requirements Engineering*, 4<sup>th</sup> ed., Springer International Publishing, Cham, Switzerland, 2017. ISBN: 978-3319-61072-6.
- [13] Ponsard, C., Massonet, P., Rifaut, A., Molderez, J. F., van Lamsweerde, A., and Tran Van, H., “Early Verification and Validation of Mission Critical Systems,” *Electronic Notes in Theoretical Computer Science*, Vol. 133, 2005, pp. 237–254. doi:10.1016/j.entcs.2004.08.067.
- [14] Friedenthal, S., Moore, A., and Steiner, R., *A Practical Guide to SysML: The Systems Modeling Language*, 1<sup>st</sup> ed., Morgan Kaufmann Publishers Inc., New York, NY, USA, 2008. ISBN: 978-0-12-374379-4.
- [15] Delligatti, L., *SysML Distilled: A Brief Guide to the Systems Modeling Language*, Addison-Wesley, 2013. ISBN: 978-0-321-92786-6.
- [16] Kaslow, D., Anderson, L., Asundi, S., Ayres, B., Iwata, C., Shiotani, B., and Thompson, R., “Developing a CubeSat Model-Based System Engineering (MBSE) Reference Model - Interim Status,” *2015 IEEE Aerospace Conference* (7-14 March 2015, Big Sky, MT, USA), 2015. doi:10.1109/AERO.2015.7118965.
- [17] Henderson, K., and Salado, A., “Value and Benefits of Model-Based Systems Engineering (MBSE): Evidence From the Literature,” *Systems Engineering*, Vol. 24, No. 1, 2021, pp. 51–66. doi:10.1002/sys.21566.
- [18] Lerat, J.-P., “Three Reasons Why Document-Based SE (Usually) Works Better Than (Most of) MBSE,” *INCOSE International Symposium*, Vol. 20, No. 1, 2010, pp. 723–738. doi:<https://doi.org/10.1002/j.2334-5837.2010.tb01100.x>.
- [19] Wymore, A. W., *Model-Based Systems Engineering*, CRC Press, Boca Raton, FL, USA, 1993.
- [20] Huldt, T., and Stenius, I., “State-of-Practice Survey of Model-Based Systems Engineering,” *Systems Engineering*, Vol. 22, No. 2, 2019, pp. 134–145. doi:10.1002/sys.21466.
- [21] Friedenthal, S., and Oster, C., *Architecting Spacecraft with SysML: A Model-Based Systems Engineering Approach*, CreateSpace Independent Publishing Platform, 2017. ISBN: 978-1-54-428806-2.
- [22] Wertz, J. R., Everett, D. F., and Puschell, J. J. (eds.), *Space Mission Engineering: The New SMAD*, 2<sup>nd</sup> ed., Microcosm Press, Hawthorne, CA, USA, 2015. ISBN: 978-1-881-883-15-9.

- [23] Kaslow, D., Ayres, B., Cahill, P. T., Hart, L., and Yntema, R., "A Model-Based Systems Engineering (MBSE) Approach for Defining the Behaviors of CubeSats," *2017 IEEE Aerospace Conference* (4-11 March 2017, Big Sky, MT, USA), 2017. doi:10.1109/AERO.2017.7943865.
- [24] Kaslow, D., Ayres, B., Cahill, P. T., and Hart, L., "A Model-Based Systems Engineering Approach for Technical Measurement with Application to a CubeSat," *2018 IEEE Aerospace Conference* (3-10 March 2018, Big Sky, MT, USA), 2018. doi:10.1109/AERO.2018.8396443.
- [25] Anderson, L., Cole, B., Yntema, R., Bajaj, M., Spangelo, S., Kaslow, D., Lowe, C., Sudano, E., Boghosian, M., Reil, R., Asundi, S., and Friedenthal, S., "Enterprise Modeling for CubeSats," *2014 IEEE Aerospace Conference* (1-8 March 2014, Big Sky, MT, USA), 2014. doi:10.1109/AERO.2014.6836343.
- [26] Spangelo, S. C., Cutler, J., Anderson, L., Fosse, E., Leo Cheng, Yntema, R., Bajaj, M., Delp, C., Cole, B., Soremekum, G., and Kaslow, D., "Model Based Systems Engineering (MBSE) Applied to Radio Aurora Explorer (RAX) CubeSat Mission Operational Scenarios," *2013 IEEE Aerospace Conference* (2-9 March 2013, Big Sky, MT, USA), 2013. doi:10.1109/AERO.2013.6496894.
- [27] Kaslow, D., Soremekun, G., Kim, H., and Spangelo, S., "Integrated Model-Based Systems Engineering (MBSE) Applied to the Simulation of a CubeSat Mission," *2014 IEEE Aerospace Conference* (1-8 March 2014, Big Sky, MT, USA), 2014. doi:10.1109/AERO.2014.6836317.
- [28] Phojanamongkolkij, N., Lee, K., Miller, S. T., Vorndran, K. A., Vaden, K. R., Ross, E. P., Powell, R. C., and Moses, R., "Modeling to Mars: a NASA Model Based Systems Engineering Pathfinder Effort," *AIAA SPACE and Astronautics Forum and Exposition* (12-14 September 2017, Orlando, FL, USA), 2017. doi:10.2514/6.2017-5235.
- [29] Eisenmann, H., "MBSE Has a Good Start; Requires More Work for Sufficient Support of Systems Engineering Activities Through Models," *INSIGHT*, Vol. 18, No. 2, 2015, pp. 14–18. doi:10.1002/inst.12014.
- [30] Schluse, M., Atorf, L., and Rossmann, J., "Experimentable Digital Twins for Model-Based Systems Engineering and Simulation-Based Development," *2017 Annual IEEE International Systems Conference (SysCon)* (24-27 April 2017, Montreal, QC, Canada), 2017, pp. 1–8. doi:10.1109/SYSCON.2017.7934796.
- [31] European Space Agency (ESA), "OCDT - Open Concurrent Design Tool," , 2021. URL <https://ocdt.esa.int/>, [Online; accessed 16-Jun-2021].
- [32] Chabibi, B., Anwar, A., and Nassar, M., "Towards a Model Integration from SysML to MATLAB/Simulink," *Journal of Software*, Vol. 13, No. 12, 2018, pp. 630–645. doi:10.17706/jsw.13.12.630-645.
- [33] Kim, H., Fried, D., and Menegay, P., "Connecting SysML Models with Engineering Analyses to Support Multidisciplinary System Development," *12th AIAA Aviation Technology, Integration, and*

- Operations (ATIO) Conference* (17-19 September 2012, Indianapolis, IN, USA), 2012. doi:10.2514/6.2012-5632.
- [34] Johnson, T., Paredis, C., and Burkhart, R., “Integrating Models and Simulations of Continuous Dynamics Into SysML,” *Journal of Computing and Information Science in Engineering*, Vol. 12, No. 1, 2012. doi:10.1115/1.4005452.
- [35] Nigischer, C., Bougain, S., Riegler, R., Stanek, H. P., and Grafinger, M., “Multi-Domain Simulation Utilizing SysML: State of the Art and Future Perspectives,” *Procedia CIRP*, Vol. 100, 2021, pp. 319–324. doi:10.1016/j.procir.2021.05.073.
- [36] Zeigler, B. P., Mittal, S., and Traore, M. K., “MBSE with/out Simulation: State of the Art and Way Forward,” *Systems*, Vol. 6, No. 4, 2018. doi:10.3390/systems6040040.
- [37] Monteiro, J. P., Gil, P. J. S., and Rocha, R. M., “A Taxonomy for Model-Based Systems Engineering,” paper presented at ASME 2021 International Design Engineering Technical Conference, 17–19 August (Online), 2021.
- [38] Madni, A. M., and Sievers, M., “Model-Based Systems Engineering: Motivation, Current Status, and Research Opportunities,” *Systems Engineering*, Vol. 21, No. 3, 2018, pp. 172–190. doi: 10.1002/sys.21438.
- [39] Chami, M., and Bruel, J., “A Survey on MBSE Adoption Challenges,” *Proceedings of the EMEASEC 2018* (5-7 November 2018, Berlin, Germany), 2018.
- [40] Nelson, T., Borky, J. M., and Sega, R. M., “System-of-Systems Quality Attribute-Based Architectural Alternatives,” *IEEE Systems Journal*, Vol. 14, No. 3, 2020, pp. 3844–3854. doi: 10.1109/JSYST.2019.2935984.
- [41] LaSorda, M., Borky, J., and Sega, R., “Model-Based Architecture and Programmatic Optimization for Satellite System-of-Systems Architectures,” *Systems Engineering*, Vol. 21, No. 4, 2018, pp. 372–387.
- [42] Spangelo, S. C., Kaslow, D., Delp, C., Cole, B., Anderson, L., Fosse, E., Gilbert, B. S., Hartman, L., Kahn, T., and Cutler, J., “Applying Model Based Systems Engineering (MBSE) to a Standard CubeSat,” *2012 IEEE Aerospace Conference* (3-10 March 2012, Big Sky, MT, USA), 2012. doi: 10.1109/AERO.2012.6187339.
- [43] Kaslow, D., Hart, L., Ayres, B., Massa, C., Chonoles, M. J., Yntema, R., Gasster, S. D., and Shiotani, B., “Developing a CubeSat Model-Based System Engineering (MBSE) Reference Model — Interim Status #2,” *2016 IEEE Aerospace Conference* (5-12 March 2016, Big Sky, MT, USA), 2016. doi:10.1109/AERO.2016.7500592.
- [44] Kaslow, D., Ayres, B., Cahill, P. T., Hart, L., and Yntema, R., “Developing a CubeSat Model-Based System Engineering (MBSE) Reference Model — Interim Status #3,” *2017 IEEE Aerospace Conference* (4-11 March 2017, Big Sky, MT, USA), 2017. doi:10.1109/AERO.2017.7943691.

- [45] Waseem, M., and Sadiq, M. U., "Application of Model-Based Systems Engineering in Small Satellite Conceptual Design – A SysML Approach," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 33, No. 4, 2018, pp. 24–34. doi:10.1109/MAES.2017.180230.
- [46] Mazzini, S., Tronci, E., Paccagnini, C., and Olive, X., "A Model-Based Methodology to Support the Space System Engineering (MBSSE)," paper presented at the Embedded Real Time Software and Systems (ERTS2) 2010 Congress, 19-21 April, Toulouse, France, 2010.
- [47] Jakob, F., Mazzini, S., and Jung, A., "A SysML-Based Methodology in a Concurrent Satellite Design Process," Tech. Rep. 2011-01-2713, Society of Automotive Engineers, 2011. doi:10.4271/2011-01-2713.
- [48] Hennig, C., Viehl, A., Kämpgen, B., and Eisenmann, H., "Ontology-Based Design of Space Systems," *The Semantic Web – ISWC 2016* (17–21 October 2016, Kobe, Japan), Springer International Publishing, 2016, pp. 308–324.
- [49] Edmonson, W., Chenou, J., Neogi, N., and Herencia-Zapana, H., "Small Satellite Systems Design Methodology: A Formal and Agile Design Process," *2014 IEEE International Systems Conference Proceedings* (31 March-3 April 2014, Ottawa, ON, Canada), 2014, pp. 518–524. doi:10.1109/SysCon.2014.6819305.
- [50] Gao, S., Cao, W., Fan, L., and Liu, J., "MBSE for Satellite Communication System Architecting," *IEEE Access*, Vol. 7, 2019, pp. 51–67. doi:10.1109/ACCESS.2019.2952889.
- [51] Guo, J., Gill, E., and Figari, S., "Model Based Systems Engineering to Support the Development of Nano-Satellites," paper presented at the 65th International Astronautical Congress, 29 September - 3 October, Toronto, Canada, 2014.
- [52] Romero, A. G., and Ferreira, M., "Modeling an attitude and orbit control system using SysML," paper presented at the 2nd Workshop in Space Technology and Engineering, 3-4 May, São Paulo, Brazil, 2011.
- [53] Garro, A., Groß, J., Riestenpatt gen. Richter, M., and Tundis, A., "Reliability Analysis of an Attitude Determination and Control System (ADCS) Through the RAMSAS Method," *Journal of Computational Science*, Vol. 5, No. 3, 2014, pp. 439–449. doi:10.1016/j.jocs.2013.06.003.
- [54] Hiep, C., and Ioki, M., "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.
- [55] Garro, A., and Tundis, A., "A Model-Based Method for System Reliability Analysis," *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium* (26-30 March 2012, Orlando, FL, USA), 2012.



- [56] Karban, R., Jankevičius, N., and Elaasar, M., "ESEM: Automated Systems Analysis Using Executable SysML Modeling Patterns," *INCOSE International Symposium*, Vol. 26, No. 1, 2016, pp. 1–24. doi:10.1002/j.2334-5837.2016.00142.x.
- [57] Cao, Y., Liu, Y., and Paredis, C. J., "System-Level Model Integration of Design and Simulation for Mechatronic Systems Based on SysML," *Mechatronics*, Vol. 21, No. 6, 2011, pp. 1063–1075. doi:10.1016/j.mechatronics.2011.05.003.
- [58] Kapos, G., Dalakas, V., Tsadimas, A., Nikolaidou, M., and Anagnostopoulos, D., "Model-Based System Engineering Using SysML: Deriving Executable Simulation Models with QVT," *2014 IEEE International Systems Conference Proceedings* (31 March-3 April 2014, Ottawa, ON, Canada), 2014, pp. 531–538. doi:10.1109/SysCon.2014.6819307.
- [59] Object Management Group (OMG), "MOF Query/View/Transformation Specification, version 1.3," , 2016. URL <https://www.omg.org/spec/QVT/About-QVT>, [Online; accessed 12-Apr-2021].
- [60] ECSS Secretariat, "Space Engineering - System Modelling and Simulation," Tech. Rep. ECSS-E-TM-10-21A, ESA-ESTEC Requirements and Standards Division, Noordwijk, The Netherlands, 2010.
- [61] Martin, J. N., *Systems Engineering Guidebook: A Process for Developing Systems and Products*, CRC Press, Boca Raton, FL, USA, 1996. ISBN: 978-0-84-937837-9.
- [62] Estefan, J., "Survey of Model-Based Systems Engineering (MBSE) Methodologies, Rev. B," Tech. rep., NASA Jet Propulsion Lab, 2008.
- [63] Forsberg, K., and Mooz, H., "The Relationship of System Engineering to the Project Cycle," *INCOSE International Symposium*, Vol. 1, No. 1, 1991, pp. 57–65. doi:10.1002/j.2334-5837.1991.tb01484.x.
- [64] Douglass, B. P., "The Harmony Process", *Real-Time UML Workshop for Embedded Systems*, 2<sup>nd</sup> ed., Elsevier Inc, 2014, pp. 33–66. ISBN: 978-0-12-407781-2.
- [65] Dori, D., *Object-Process Methodology: A Holistic Systems Paradigm*, Springer-Verlag, Berlin, Germany, 2002. ISBN: 978-3-642-62989-1.
- [66] Object Management Group (OMG), "OMG Systems Modeling Language Specification, version 1.6," , 2019. URL <https://www.omg.org/spec/SysML/1.6/About-SysML/>, [Online; accessed 12-Apr-2021].
- [67] Object Management Group (OMG), "SysML Extension for Physical Interaction and Signal Flow Simulation, version 1.1," , 2018. URL <https://www.omg.org/spec/SysPhS/About-SysPhS/>, [Online; accessed 12-Apr-2021].
- [68] Royce, W. W., "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings of the 9th International Conference on Software Engineering* (March 1987, Monterey, CA, USA), 1987, p. 328–338.

- [69] Boehm, B. W., "A Spiral Model of Software Development and Enhancement," *Computer*, Vol. 21, No. 5, 1988, p. 61–72. doi:10.1109/2.59.
- [70] ECSS Secretariat, "Space Engineering - Control Engineering Handbook," Tech. Rep. ECSS-E-HB-60A, ESA-ESTEC Requirements and Standards Division, Noordwijk, The Netherlands, 2010.
- [71] Markley, F. L., and Crassidis, J. L., *Fundamentals of Spacecraft Attitude Determination and Control*, 1<sup>st</sup> ed., Springer, New York, NY, USA, 2014. ISBN: 978-1-4939-0801-1.
- [72] Gaber, K., Nagy, S., and Zaki, A., "MEMS Gyroscope for Attitude Propagation and Determination for Nanosatellites," *2017 Intl Conf on Advanced Control Circuits Systems (ACCS) Systems and 2017 Intl Conf on New Paradigms in Electronics and Information Technology (PEIT)* (5-8 November 2017, Alexandria, Egypt), 2017. doi:10.1109/ACCS-PEIT.2017.8303036.
- [73] Xia, X., Sun, G., Zhang, K., Wu, S., Wang, T., Xia, L., and Liu, S., "NanoSats/CubeSats ADCS Survey," *2017 29th Chinese Control And Decision Conference (CCDC)* (28-30 May 2017, Chongqing, China), 2017. doi:10.1109/CCDC.2017.7979410.
- [74] Crassidis, J. L., Markley, F. L., and Cheng, Y., "Survey of Nonlinear Attitude Estimation Methods," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 1, 2007, pp. 12–28. doi:10.2514/1.22452.
- [75] Lefferts, E. J., Markley, F. L., and Shuster, M. D., "Kalman Filtering for Spacecraft Attitude Estimation," *Journal of Guidance, Control, and Dynamics*, Vol. 5, No. 5, 1982, pp. 417–429. doi:10.2514/3.56190.
- [76] Raymond, K., Egeland, O., and Nicklasson, P., "A Comparative Study of Actuator Configurations for Satellite Attitude Control," *Modeling, Identification and Control*, Vol. 26, No. 4, 2005, pp. 201–220. doi:10.4173/mic.2005.4.2.
- [77] Wen, J.-Y., and Kreutz-Delgado, K., "The Attitude Control Problem," *IEEE Transactions on Automatic Control*, Vol. 36, No. 97, 1991, pp. 1148–1162. doi:10.1109/9.90228.
- [78] Tassano, M., Monzon, P., and Pechiar, J., "Attitude Determination and Control System of the Uruguayan Cubesat, AntelSat," *2013 16th International Conference on Advanced Robotics (ICAR)* (25-29 November 2013, Montevideo, Uruguay), 2014. doi:10.1109/ICAR.2013.6766523.
- [79] Brum, A., Baroni, L., da Silva, A., Coelho, F., Ferreira, E., Zanardi, M., and Spengler, A., "Attitude Control System Proposed for SERPENS-2 Space Mission," *Computational and Applied Mathematics*, Vol. 37, 2018, pp. 1–14. doi:10.1007/s40314-018-0574-x.
- [80] Ran, D., Sheng, T., Cao, L., Chen, X., and Zhao, Y., "Attitude Control System Design and On-Orbit Performance Analysis of Nano-Satellite – "Tian Tuo 1"," *Chinese Journal of Aeronautics*, Vol. 27, No. 3, 2014, pp. 593–601. doi:10.1016/j.cja.2013.11.001.

- [81] ECSS Secretariat, "Space Engineering - Control Performance Guidelines," Tech. Rep. ECSS-E-HB-60-10A, ESA-ESTEC Requirements and Standards Division, Noordwijk, The Netherlands, 2010.
- [82] The MathWorks Inc., "Stateflow API," , 2021. URL [https://www.mathworks.com/help/pdf\\_doc/stateflow/stateflow\\_api.pdf](https://www.mathworks.com/help/pdf_doc/stateflow/stateflow_api.pdf), [Online; accessed 12-Abr-2021].
- [83] The CubeSat Program, Cal Poly SLO, "CubeSat Design Specification, rev 13," , February 2014. URL [https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds\\_rev13\\_final2.pdf](https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf).
- [84] Monteiro, J. P., Rocha, R. M., Silva, A., Afonso, R., and Ramos, N., "Integration and Verification Approach of ISTSat-1 CubeSat," *Aerospace*, Vol. 6, No. 12, 2019. doi:10.3390/aerospace6120131.
- [85] Neves, D., "Control Algorithm for ISTSat-1," Master's thesis, Instituto Superior Técnico, University of Lisbon, Portugal, 2019.
- [86] ECSS Secretariat, "Space Engineering - Satellite Attitude and Orbit Control System (AOCS) Requirements," Tech. Rep. ECSS-E-ST-60-30C, ESA-ESTEC Requirements and Standards Division, Noordwijk, The Netherlands, 2013.
- [87] CubeSpace, "CubeSense V3 - Interface Control Document, version 1.3," , 2021. URL [https://drive.google.com/uc?export=download&id=1KXZ0Gk\\_GKfY-Irtgx3fAJLdsIHjFTzUG](https://drive.google.com/uc?export=download&id=1KXZ0Gk_GKfY-Irtgx3fAJLdsIHjFTzUG), [Online; accessed 12-Oct-2021].
- [88] InvenSense, "MPU-9250 Product Specification, Revision 1.1," , 2016. URL <https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf>, [Online; accessed 12-Oct-2021].
- [89] Honeywell, "3-Axis Digital Compass IC HMC5983," , 2011. URL [http://www.farnell.com/datasheets/1509871.pdf?\\_ga=2.219060057.1318745487.1555987311-293789508.1555987311](http://www.farnell.com/datasheets/1509871.pdf?_ga=2.219060057.1318745487.1555987311-293789508.1555987311), [Online; accessed 12-Oct-2021].
- [90] CubeSpace, "CubeWheel - Interface Control Document, version 1.17," , 2021. URL [https://drive.google.com/uc?export=download&id=13ZzUQH0uU1pjZ86910C3\\_PQYGs03g0I1](https://drive.google.com/uc?export=download&id=13ZzUQH0uU1pjZ86910C3_PQYGs03g0I1), [Online; accessed 12-Oct-2021].
- [91] EnduroSat, "1U Solar Panel – Specification," , 2021. URL <https://www.endurosat.com/cubesat-store/cubesat-solar-panels/1u-solar-panel-x-y/>, [Online; accessed 12-Oct-2021].
- [92] Chulliat, A., Brown, W., Alken, P., Beggan, C., Nair, M., Cox, G., Woods, A., Macmillan, S., Meyer, B., and Paniccia, M., "The US/UK World Magnetic Model for 2020-2025," Tech. rep., National Centers for Environmental Information, NOAA, 2020. doi:10.25923/ytk1-yx35.

- [93] Bhanderi, D., and Bak, T., "Modeling Earth Albedo for Satellites in Earth Orbit," *Proceedings of AIAA Conference on Guidance, Navigation and Control* (15-18 August 2005, San Francisco, CA, USA), 2005. doi:10.2514/6.2005-6465.
- [94] GMAT Development Team, "General Mission Analysis Tool (GMAT) [Computer software]," Tech. rep., NASA Goddard Space Flight Center, 2017. URL <http://sourceforge.net/projects/gmat>.
- [95] Montenbruck, O., and Gill, E., *Satellite Orbits: Models, Methods and Applications*, Springer-Verlag, Berlin, Germany, 2000. ISBN: 978-3-540-62780-7.
- [96] Standish, E. M., "JPL Planetary and Lunar Ephemerides, DE405/LE405," , 1998. JPL Interoffice Memorandum 312.F-98-048.
- [97] Dekker, L., "Simulation Environments," *Systems Analysis and Simulation II* (12–16 September 1988, Berlin, Germany), Springer US, 1988, pp. 344–350. doi:10.1007/978-1-4613-8936-1\_74.
- [98] Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, 4<sup>th</sup> ed., Microcosm Press, Hawthorne, CA, USA, 2013. ISBN: 978-1881883180.
- [99] Petit, G., and Luzum, B. (eds.), *IERS Conventions (2010), Technical Note No. 36*, International Earth Rotation and Reference Systems Service (IERS), 2010. URL <https://www.iers.org/SharedDocs/Publikationen/EN/IERS/Publications/tn/TechnNote36/tn36.html>.
- [100] Wie, B., *Space Vehicle Dynamics and Control*, 2<sup>nd</sup> ed., American Institute of Aeronautics and Astronautics Inc., Reston, VA, USA, 2008. ISBN: 978-1-56347-953-3.
- [101] ECSS Secretariat, "Space Engineering - Control Performance," Tech. Rep. ECSS-E-ST-60-10C, ESA-ESTEC Requirements and Standards Division, Noordwijk, The Netherlands, 2008.
- [102] Pavlis, N. K., Holmes, S. A., Kenyon, S. C., and Factor, J. K., "The Development and Evaluation of the Earth Gravitational Model 2008 (EGM2008)," *Journal of Geophysical Research: Solid Earth*, Vol. 117, No. B4, 2012. doi:10.1029/2011JB008916.
- [103] Bhanderi, D. D. V., "Earth Albedo Toolbox v1.0," , 2021. URL <https://bhanderi.dk/>, [Online; accessed 05-Jul-2021].
- [104] Lam, Q., Stamatakos, N., Woodruff, C., and Ashton, S., "Gyro Modeling and Estimation of Its Random Noise Sources," *AIAA Guidance, Navigation, and Control Conference and Exhibit* (11-14 August 2003, Austin, TX, USA), 2003. doi:10.2514/6.2003-5562.
- [105] Mahony, R., Hamel, T., and Pflimlin, J.-M., "Nonlinear Complementary Filters on the Special Orthogonal Group," *IEEE Transactions on Automatic Control*, Vol. 53, No. 5, 2008, pp. 1203–1218. doi:10.1109/TAC.2008.923738.
- [106] Utstumo, T., and Gravdahl, J. T., "Implementation and Comparison of Attitude Estimation Methods for Agricultural Robotics," *IFAC Proceedings Volumes*, Vol. 46, No. 18, 2013, pp. 52–57. doi: 10.3182/20130828-2-SF-3019.00051.

- [107] Avanzini, G., and Giulietti, F., “Magnetic Detumbling of a Rigid Spacecraft,” *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 4, 2012, pp. 1326–1334. doi:10.2514/1.53074.



# Appendix A

## Spacecraft Attitude

### A.1 Reference Frames

In this section, different reference frames are discussed with support from literature [71, 98], all of which are right-handed and orthogonal.

#### A.1.1 Inertial Reference Frame

Reference frames fixed relative to distant stars are currently the best realizations of inertial frames [71]. The standard at this moment is the International Celestial Reference Frame (ICRF), which is fixed with respect to distant extragalactic sources of radio waves. The origin of the ICRF is the centre of mass of the solar system, the x axis is aligned with the *vernal equinox*, and the z axis is aligned with the Earth's North pole. In turn, the *vernal equinox* is aligned with the intersection of the Earth's equatorial plane with the ecliptic plane, and it is in the direction of the Sun's position relative to Earth at the moment of the vernal equinox (start of spring). Since the Earth's polar axis and the ecliptic plane are not inertially fixed, the x and z axes of the ICRF are actually aligned with the mean orientations of the *vernal equinox* and the Earth's North pole, respectively, at some fixed epoch time.

The Geocentric Celestial Reference Frame (GCRF) is the geocentric counterpart of the ICRF [98]. This coordinate frame is not inertial relative to the ICRF because it has a linear acceleration that results from the Earth's orbit around the Sun. However, in the context of attitude analysis, it can be assumed to be approximately inertial [71]. In this work, the reference frame used for the integration of the spacecraft position is the Earth-Centred Inertial (ECI) frame, which in this work is defined to be equal to the realization of the GCRF at J2000. The ECI frame is denoted by the letter  $I$  such that the representation of a vector  $u$  in this frame is shown as  $u_I$ .

#### A.1.2 Earth-Centred/Earth-Fixed Frame

The International Terrestrial Reference Frame (ITRF) is a geocentric coordinate system fixed to the rotating Earth. Its origin is located at the centre of mass of Earth, the z axis is aligned with Earth's North

pole, and the x axis is aligned with the Earth's prime meridian. To compute vector quantities that are fixed relative to Earth, we use the Earth-Centred/Earth-Fixed (ECEF) frame, which in this work corresponds to the ITRF realized by the IAU2000/2006 reduction from the ICRF. The ECEF frame is denoted by the letter  $E$  such that the representation of a vector  $u$  in this frame is shown as  $u_E$ .

The transformation between ECI and ECEF using the IAU2000/2006 reduction is quite complex and can be consulted in [99]. Applying this transformation at the defined time of separation  $T_0$  (1 January 2022, 12:00:00 UTC), the transformation of a position vector  $r$  from its representation in the ECI frame  $r_I$  to its representation in the ECEF frame realized at the time of separation  $r_{E'}$  is as follows:

$$r_{E'} = C_{EI}(t = T_0) r_I = \begin{bmatrix} 0.19239280 & -0.98131792 & -3.8507196 \times 10^{-4} \\ 0.98131593 & 0.19239318 & -1.9722281 \times 10^{-3} \\ 2.00946808 \times 10^{-3} & 1.56525356 \times 10^{-6} & 0.9999979 \end{bmatrix} r_I. \quad (\text{A.1})$$

Performing a coordinate transformation from ECI to ECEF with the IAU2000/2006 reduction is too complex for the estimation algorithms. Therefore, a simpler approach is taken, which consists of using two separate transformations, the first from the ECI frame to the ECEF frame realized at  $T_0$ , and the second from the ECEF frame realized at  $T_0$  to the ECEF realized at the instant relevant to the algorithm. The difference between these two instants in seconds is denoted as  $\Delta T$ . The same type of process is applied to transformations from ECEF to ECI.

The first transformation is independent of time and is expressed in Eq. A.1. The second transformation is simpler, and it considers that the ECEF frame rotates relative to the ECI frame around its z axis, with a constant angular speed. The rotation angle of the ECEF frame relative to the ECI frame is known as the Greenwich Mean Sidereal Time (GMST) angle and it is denoted as  $\theta_{\text{GMST}}$ . The time derivative of this angle  $\dot{\theta}_{\text{GMST}}$  is defined as

$$\dot{\theta}_{\text{GMST}} = \omega_{\oplus}, \quad (\text{A.2})$$

where  $\omega_{\oplus} = 2\pi/86164 \text{ rad s}^{-1}$  is the Earth's angular rotation speed. The second transformation is then given by

$$r_E = C_{EE'}(\Delta T) r_{E'} = \begin{bmatrix} \cos(\omega_{\oplus}\Delta T) & \sin(\omega_{\oplus}\Delta T) & 0 \\ -\sin(\omega_{\oplus}\Delta T) & \cos(\omega_{\oplus}\Delta T) & 0 \\ 0 & 0 & 1 \end{bmatrix} r_{E'}. \quad (\text{A.3})$$

The ECEF position is also represented in geocentric spherical coordinates, specified by its magnitude  $r \equiv ||r_E||$  and two angles, longitude  $\lambda$  and geocentric latitude  $\phi'$ , which relate to ECEF cartesian coordinates ( $r_E \equiv \{x_E, y_E, z_E\}$ ) as follows:

$$\lambda = \tan^{-1} \left( \frac{x_E}{y_E} \right), \quad (\text{A.4a})$$

$$\phi' = \sin^{-1}(z_E/r). \quad (\text{A.4b})$$

The geodetic spherical coordinates are not used in this work. These differ from the previous in terms of the latitude definition (geodetic latitude is denoted as  $\phi$ ).



### A.1.3 Topocentric Horizon Coordinate System (NED)

In this work we use the North-East-Down (NED) coordinate system for the estimation of the Earth local magnetic field vector and acceleration of gravity. This system is consistent with the Topocentric Horizon Coordinate System defined in [98], also called South-East-Zenith (SEZ). It corresponds to the SEZ frame rotated 180 degrees around the E axis. The origin of the frame is located at the centre of mass of the spacecraft, the x axis points due north, the y axis points due east, and the z axis points radially inward towards the centre of Earth. The NED frame is denoted by the letter  $H$  such that the representation of a vector  $\mathbf{u}$  in this frame is shown as  $\mathbf{u}_H$ .

If the position of the spacecraft is known in geocentric spherical coordinates, the transformation of a vector  $\mathbf{u}$  represented in this frame into the ECEF frame representation is given by

$$\mathbf{u}_E = C_{EH} \mathbf{u}_H = \begin{bmatrix} -\cos(\lambda) \sin(\phi') & -\sin(\lambda) & -\cos(\lambda) \cos(\phi') \\ -\sin(\lambda) \sin(\phi') & \cos(\lambda) & -\sin(\lambda) \cos(\phi') \\ \cos(\phi') & 0 & -\sin(\phi') \end{bmatrix} \mathbf{u}_H. \quad (\text{A.5})$$

### A.1.4 Spacecraft Body Frame

The spacecraft body frame is fixed relative to the spacecraft and the orientation of its axes is usually defined by the engineers that design the system. For CubeSats, the origin of the frame is usually located at the spacecraft's centre of mass and the axes are usually orthogonal to the faces of the satellite, as shown in Figure A.1. Also shown in this figure is the nomenclature used to describe each face of the spacecraft (e.g.,  $X^+$  or  $Z^-$ ). The nomenclature corresponds to a letter representing the orthogonal axis followed by a sign representing on which side of the axis the face is located.

Sensor measurements and control torques are expressed in this frame. For simplicity, we consider that all the sensors and actuators are correctly aligned with the axes of the frame. We denote the body frame by the letter  $B$  so that the representation of a vector  $\mathbf{u}$  in this frame is shown as  $\mathbf{u}_B$ .

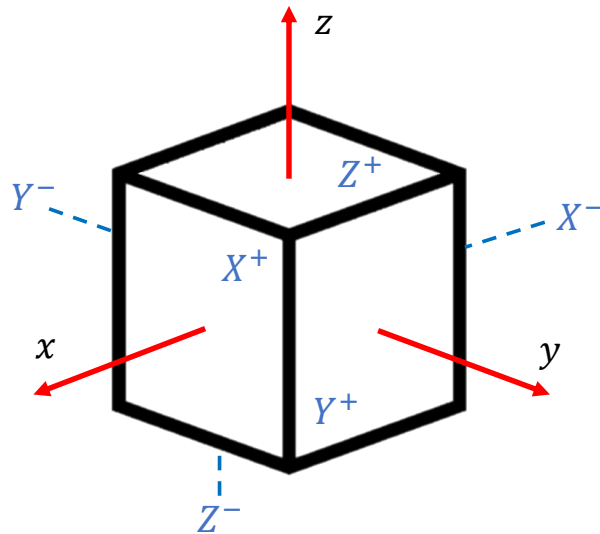


Figure A.1: Spacecraft body frame and nomenclature used to describe faces.

## A.2 Attitude Representation

### A.2.1 Attitude Matrix

Let us consider two reference frames  $F$  and  $F'$  with right-handed orthogonal bases  $\{e_1, e_2, e_3\}$  and  $\{e'_1, e'_2, e'_3\}$ , respectively, represented in the same frame. To transform the representation of the vector  $u$  in frame  $F'$  into the representation in frame  $F$  we use the equation [100]

$$u_F = C_{FF'} u_{F'}, \quad (\text{A.6})$$

where  $C_{FF'} \equiv [C_{ij}]$  is called the direction cosine matrix (DCM) or rotation matrix and  $C_{ij} \equiv e_i \cdot e'_j$ . Determining the spacecraft attitude consists of estimating a rotation matrix that transforms vectors from a fixed reference frame, in this case the ECI frame, to a frame fixed relative to the spacecraft body [71, p. 40]. This rotation matrix is usually called an attitude matrix and is denoted by the letter  $A$ , such that, for any given vector  $u$ ,

$$u_B = A_{BI} u_I. \quad (\text{A.7})$$

The attitude of a body can also be represented by the rotation around a fixed axis. Euler's Theorem states that every rotation is a rotation about a fixed axis called the Euler axis (represented in this work by the vector  $a_e$ ), with a certain angle of rotation, the Euler angle (denoted here by  $\nu$ ). Different methods for attitude representation are reviewed in literature [71], and for this thesis the quaternion was selected.

### A.2.2 Quaternions

A quaternion is a mathematical concept that was first introduced by William Rowan Hamilton in 1844. In this work we adopted the convention from [71], which differs from the originally introduced by Hamilton. A quaternion  $q$  may be represented as a vector, which is composed by a three-dimensional vector  $q_{1:3} = \{q_1, q_2, q_3\}$  and a scalar  $q_4$ , so that

$$q = \begin{bmatrix} q_{1:3} \\ q_4 \end{bmatrix}. \quad (\text{A.8})$$

Along with this representation, there are a set of definitions regarding the conjugate, norm, and inverse, which are respectively given by:

$$q^* = \begin{bmatrix} -q_{1:3} \\ q_4 \end{bmatrix}; \quad (\text{A.9a})$$

$$\|q\| = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}; \quad (\text{A.9b})$$

$$q^{-1} = \frac{q^*}{\|q\|^2}. \quad (\text{A.9c})$$

There are also two quaternion product operations important in this context. For two arbitrary quater-

nions  $\mathbf{q}$  and  $\mathbf{p}$ , these are defined as

$$\mathbf{q} \otimes \mathbf{p} = \begin{bmatrix} p_4 \mathbf{q}_{1:3} + q_4 \mathbf{p}_{1:3} - \mathbf{q}_{1:3} \times \mathbf{p}_{1:3} \\ q_4 p_4 - \mathbf{q}_{1:3} \cdot \mathbf{p}_{1:3} \end{bmatrix} = \mathbf{p} \odot \mathbf{q} \quad (\text{A.10})$$

$$\mathbf{q} \odot \mathbf{p} = \begin{bmatrix} q_4 \mathbf{p}_{1:3} + p_4 \mathbf{q}_{1:3} - \mathbf{p}_{1:3} \times \mathbf{q}_{1:3} \\ p_4 q_4 - \mathbf{p}_{1:3} \cdot \mathbf{q}_{1:3} \end{bmatrix} \quad (\text{A.11})$$

These products are associative and distributive but in general not commutative. The quaternion products may be represented by matrix multiplication such that

$$\mathbf{q} \otimes \mathbf{p} = [\mathbf{q} \otimes] \mathbf{p} \quad (\text{A.12})$$

and

$$\mathbf{q} \odot \mathbf{p} = [\mathbf{q} \odot] \mathbf{p}, \quad (\text{A.13})$$

where

$$[\mathbf{q} \otimes] = \begin{bmatrix} q_4 I_3 - [\mathbf{q}_{1:3} \times] & \mathbf{q}_{1:3} \\ -\mathbf{q}_{1:3}^T & q_4 \end{bmatrix} = \begin{bmatrix} \Psi(\mathbf{q}) & \mathbf{q} \end{bmatrix} \quad (\text{A.14})$$

and

$$[\mathbf{q} \odot] = \begin{bmatrix} q_4 I_3 + [\mathbf{q}_{1:3} \times] & \mathbf{q}_{1:3} \\ -\mathbf{q}_{1:3}^T & q_4 \end{bmatrix} = \begin{bmatrix} \Xi(\mathbf{q}) & \mathbf{q} \end{bmatrix}, \quad (\text{A.15})$$

with  $[\mathbf{u}_\times]$  representing the cross product matrix for a given vector  $\mathbf{u} = \{u_1, u_2, u_3\}$ , which is defined as

$$[\mathbf{u}_\times] = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix}. \quad (\text{A.16})$$

The quaternion representation of attitude uses unit quaternions to parameterize rotations ( $\|\mathbf{q}\| = 1$ ). When the quaternion  $\mathbf{q}$  and attitude matrix  $A$  are used to represent the attitude of the spacecraft in this work, the default definitions used are  $\mathbf{q} \equiv \mathbf{q}_{BI}$  and  $A \equiv A_{BI}$ . The attitude quaternion is defined as

$$\mathbf{q}(\mathbf{a}_e, \nu) = \begin{bmatrix} \mathbf{a}_e \sin(\nu/2) \\ \cos(\nu/2) \end{bmatrix}, \quad (\text{A.17})$$

where  $\mathbf{a}_e$  is represented in the ECI frame and  $\mathbf{q}$  represents the rotation of the ECI frame around  $\mathbf{a}_e$ . Since this is a unit quaternion, the inverse and conjugate quaternions are equal and represent the inverse rotation. The attitude matrix can be obtained from the attitude quaternion with

$$A \equiv C_q(\mathbf{q}) = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1 q_2 + q_3 q_4) & 2(q_1 q_3 - q_2 q_4) \\ 2(q_2 q_1 - q_3 q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2 q_3 + q_1 q_4) \\ 2(q_3 q_1 + q_2 q_4) & 2(q_3 q_2 - q_1 q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix}. \quad (\text{A.18})$$

To compute the inverse transformation, the attitude matrix  $A_{IB} \equiv C_q(\mathbf{q}^*)$  is used. The identity quaternion  $I_q$  represents a null rotation and is defined as

$$I_q \equiv \begin{bmatrix} 0_3 \\ 1 \end{bmatrix}. \quad (\text{A.19})$$

### A.3 Spacecraft Attitude Dynamics

The fundamental equations of motion for rotational dynamics are called the Euler equations and they define how the spacecraft attitude changes over time. These equations, represented in vector form and in the spacecraft body frame, can be defined as

$$\dot{\mathbf{H}} = \mathbf{T} - \boldsymbol{\omega} \times \mathbf{H}, \quad (\text{A.20})$$

where  $\mathbf{H}$  is the angular momentum,  $\mathbf{T}$  is the sum of the external torques, and  $\boldsymbol{\omega}$  is the spacecraft angular velocity. We may relate the angular velocity with the angular momentum using the equation

$$\mathbf{H} = I_s \boldsymbol{\omega} + \mathbf{h}, \quad (\text{A.21})$$

where  $I_s$  is the moment of inertia and  $\mathbf{h}$  is the angular momentum stored by rotating objects, such as momentum wheels. Combining the two previous equations and rearranging the terms we obtain

$$I_s \dot{\boldsymbol{\omega}} = \mathbf{T} - \dot{\mathbf{h}} - \dot{I}_s \boldsymbol{\omega} - \boldsymbol{\omega} \times I_s \boldsymbol{\omega}. \quad (\text{A.22})$$

In this work the moment of inertia of the spacecraft is assumed to be constant for simplicity ( $\dot{I}_s = 0$ ). Using the quaternion representation, the attitude kinematics is given by [71]

$$\dot{\mathbf{q}} = \frac{1}{2} \begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} \otimes \mathbf{q} = \frac{1}{2} \begin{bmatrix} \Xi(\mathbf{q}) & \mathbf{q} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} = \frac{1}{2} \Xi(\mathbf{q}) \boldsymbol{\omega} = \frac{1}{2} \begin{bmatrix} q_4 I_3 + [\mathbf{q}_{1:3} \times] \\ -\mathbf{q}_{1:3}^T \end{bmatrix} \boldsymbol{\omega}, \quad (\text{A.23})$$

and the spacecraft attitude may then be integrated using the equation

$$\mathbf{q} = \mathbf{q}_0 + \int \dot{\mathbf{q}} \, dt. \quad (\text{A.24})$$

#### A.3.1 External disturbance torques

External torques include both undesirable disturbance torques and torques applied for control. In this section we discuss the sources of the disturbance torques that were considered. The Solar Radiation Pressure (SRP) torque was considered to be dominated by the aerodynamic torque in LEO [71].

**Gravity-Gradient Torque** Any nonsymmetrical rigid body in a gravity field is subject to a gravity-gradient torque. We assume that the gravity field is spherically symmetric, for simplicity, which results in

the following expression for the calculation of the gravity-gradient torque around the centre of mass [71]

$$\mathbf{T}_{\text{gg}} = \frac{3\mu_{\oplus}}{r^3} \mathbf{n} \times (I_s \mathbf{n}), \quad (\text{A.25})$$

where  $\mathbf{n} \equiv \mathbf{n}_B$  is the body frame representation of a nadir-pointing unit vector.

**Magnetic Torque** In LEO, the Earth's magnetic field intensity is high enough for undesired magnetic dipoles on-board to result into disturbance torques that can be calculated as

$$\mathbf{T}_m = \mathbf{m}_u \times \mathbf{B}_B, \quad (\text{A.26})$$

where  $\mathbf{m}_u$  is the undesired magnetic dipole and  $\mathbf{B}_B$  is the local magnetic field vector, both in the body frame.

**Aerodynamic Torque** The aerodynamic torque can be computed by modelling the spacecraft as a collection of  $N$  plates of area  $S_i$ , each with an outward normal unit vector  $\mathbf{n}_s^i$  and a centre of pressure located in  $\mathbf{r}_p^i$ . For CubeSats, these plates correspond to the six faces of the cube-shaped structure. Considering the position and velocity of the spacecraft in the ECI frame to be  $\mathbf{r}_I \equiv \{x, y, z\}$  and  $\mathbf{v}_I \equiv \{\dot{x}, \dot{y}, \dot{z}\}$ , respectively, the velocity of the spacecraft relative to the air in the body frame is given by [71]

$$\mathbf{v}_{\text{rel } B} = A \begin{bmatrix} \dot{x} + \omega_{\oplus} y \\ \dot{y} - \omega_{\oplus} x \\ \dot{z} \end{bmatrix}, \quad (\text{A.27})$$

where  $\omega_{\oplus}$  represents the angular speed of the atmosphere. Each of the plate's normal vectors makes an angle  $\theta_{\text{aero}}^i$  with the relative velocity, such that  $\cos \theta_{\text{aero}}^i = \mathbf{n}_s^i \cdot \mathbf{v}_{\text{rel } B} / \|\mathbf{v}_{\text{rel}}\| < 0$  means that the plate is on the trailing side of the spacecraft. Considering this and using a drag coefficient  $C_D$  that is empirically determined, the aerodynamic force applied in the  $i^{\text{th}}$  plate is given by [71]

$$\mathbf{F}_{\text{aero}}^i = -\frac{1}{2} \rho C_D \|\mathbf{v}_{\text{rel}}\| \mathbf{v}_{\text{rel } B} S_i \max(\cos \theta_{\text{aero}}^i, 0), \quad (\text{A.28})$$

where  $\rho$  is the atmospheric density at the height in question. The total aerodynamic torque is given by

$$\mathbf{T}_{\text{aero}} = \sum_{i=1}^N \mathbf{r}_p^i \times \mathbf{F}_{\text{aero}}^i. \quad (\text{A.29})$$

## Appendix B

# ADCS Requirements Specification

### B.1 Specification of Performance Requirements

In this section, we present a set of definitions that support the specification of ADCS performance requirements. These definitions are based on what is defined by the ECSS, specifically in [81, 101].

First, a distinction between extrinsic and intrinsic performance properties must be defined. The extrinsic performance properties are those that depend on the interaction between the system and the external environment, such as the pointing error of the ADCS. To assess these performance properties, the system environment must be modelled or at least characterized (e.g., disturbance magnitudes). The intrinsic performance properties do not depend on the interaction of the system with the exterior (e.g., stability margins). In this work we consider only the extrinsic performance properties of the SOI since the existence of an interaction between the system and the environment makes these more relevant in the context of integrating MBSE with simulation.

The extrinsic performance requirements may concern the actual performance of a system or the knowledge of that performance. The first type concerns the performance error, the difference between the target (desired) output of the system and the actual output of the system, and the second type concerns the knowledge error, the difference between the known (estimated) output of the system and the actual output of the system [101]. A performance requirement, whichever the type, is expressed mathematically by [81]

$$P(I(e_p) < I_{\max}) \geq P_C, \quad (\text{B.1})$$

where an upper bound  $I_{\max}$  is defined for the error index  $I(e_p)$ , together with a probability  $P_C$  that the error index stays below that bound. This mathematical definition of a performance requirement aids in the identification of the elements that compose the performance requirement, which are the following:

1. An error function,  $e_p$ .
2. An index to be applied to the function,  $I(e_p)$ .
3. An upper bound for the index,  $I_{\max}$ .

4. The required probability that the index stays below the upper bound,  $P_C$ .
5. A statistical interpretation for the probability  $P_C$ .

These elements consist of the following:

**Error function** The error function defines a mathematical expression for the calculation of the error, for given values of the target/estimated output and the actual output. For the ADCS attitude knowledge error, we first define an error quaternion  $\tilde{\mathbf{q}}$  such that

$$\mathbf{q}^{\text{true}} = \tilde{\mathbf{q}} \otimes \hat{\mathbf{q}}, \quad (\text{B.2})$$

and then, following the Euler's theorem, we define the error function to be the rotation angle  $\nu$  between the body frame and the estimated body frame such that

$$e_p = \nu = \cos^{-1}(\tilde{q}_4). \quad (\text{B.3})$$

For the ADCS pointing performance error, because the goal is to orientate a vector  $\mathbf{n}_{\text{pld}}$  with the local nadir, the error function consists of calculating the angle between these two unit vectors as follows

$$e_p = \cos^{-1}(\mathbf{n}_{\text{pld}} \cdot \mathbf{n}_B). \quad (\text{B.4})$$

**Error index and Probability** As defined in [101], the error index is a parameter that isolates a particular aspect of the time variation of the error. A few different error indices are presented in [81], and we use one of this for both the performance and knowledge errors. The indices are named absolute performance error (APE) and absolute knowledge error (AKE) and they are defined as the instantaneous value of the error function. For a specified interval of time, the instantaneous error is evaluated at each moment in time (separated by a small step size), and the probability  $P_C$  of meeting the requirement is equal to the fraction of instances in which the error was below an upper bound that is defined in the requirement description. This probability is defined in percentage as a confidence level.

**Statistical Interpretation** Another important aspect relative to the definition of the probability  $P_C$  is its statistical interpretation. Let us consider a probability of meeting a requirement expressed by confidence level of  $x\%$  and a set of possible system configurations represented by different combinations of time-independent parameter values. A temporal interpretation of a requirement is that the error must be below bounds for  $x\%$  of the time, in any possible system configuration. Instead, an ensemble interpretation is that the error must be below bounds at all times, for  $x\%$  of the possible system configurations [81]. Other possible interpretations exist, some of which are outlined in [81], and for this work we selected the temporal interpretation for all the requirements.

The selection of one statistical interpretation of the probability also influences the definition of an appropriate simulation campaign that supports the verification of the defined requirements. The temporal interpretation of a requirement requires that engineers be able to ensure compliance for all the different system configurations. When, for a set of time-independent parameters, a worst case can be clearly defined and justified, the verification of the requirement for the worst case will result in the verification for all cases. The alternative is time-consuming and requires the exploration of the design space. This can be achieved with a complete sweep of the design space, or with smarter techniques like Monte Carlo simulation campaigns. Sometimes these two alternatives can also be mixed, with some parameters being defined for a worst-case scenario and the other parameters being part of a design space that is explored. Since the focus of this work is not the simulation techniques used, for simplicity, we will consider that a worst-case scenario can be defined for all the time-independent parameters.

In this work, a Technical Performance Measures (TPM) consists of a scalar parameter that is directly constrained by a performance requirement. In general, the value of this parameter can be computed by processing simulation results in accordance with the definition of the performance requirement. Using the mathematical definition in B.1, a TPM is mathematically defined with the expression

$$P(I(e_p) < I_{\text{TPM}}) = P_C, \quad (\text{B.5})$$

where  $I_{\text{TPM}}$  represents the TPM. The constraint defined by a performance requirement is  $I_{\text{TPM}} < I_{\text{max}}$ .

## B.2 Complete List of ADCS Requirements

| <b>Id</b> | <b>Functional Req.</b>     | <b>Requirement Description</b>                                                                                                                                                                       |
|-----------|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7.1.1     | ADCS Modes                 |                                                                                                                                                                                                      |
| 7.1.1.1   | Safe Pointing              | In case the spacecraft transitions to <i>safe</i> mode, the ADCS shall transition to a <i>safe pointing</i> mode, in which only the magnetic torquers may be used as actuators.                      |
| 7.1.1.2   | Mode Transition Triggering | Mode transitions shall be triggered by ground request or autonomously on-board, after checking a transition condition.                                                                               |
| 7.1.1.3   | Mode Mapping               | A mapping of ADCS modes into satellite modes shall be described, to clarify all the possible configurations in which the satellite can be in relation with the ADCS.                                 |
| 7.1.1.4   | Autonomous Mode Transition | The capability shall be provided to inhibit or to force autonomous mode transitions by ground request.                                                                                               |
| 7.1.2     | Transition to Pointing     | The ADCS shall provide the capability to transition from the initial attitude and rate after launcher separation to the final mission pointing, in a safe manner, using a detumbling algorithm.      |
| 7.1.3     | Ancillary Data             | The ADCS shall process and deliver, at the frequency specified by the mission, the attitude, angular velocity, and orbit related information to other on-board functions (including eclipse status). |
| 7.1.4     | ADCS Mass                  | The ADCS shall have a total mass below 100 grams.                                                                                                                                                    |

Table B.1: ADCS Functional Requirements.



| <b>Id</b> | <b>Operational Req.</b>    | <b>Requirement Description</b>                                                                                                                                                                 |
|-----------|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7.2.1     | Design Parameter Update    | Upon ground request, the flight software shall provide the capability for in-flight update of the ADCS design parameters.                                                                      |
| 7.2.2     | Periodic Parameter Updates | The parameters requiring periodical update shall be identified and maximum update periods shall be defined and shall be greater than the maximum period of no contact with the ground station. |
| 7.2.3     | Detumbling                 | Once the launcher separation occurs, the ADCS shall ensure that the rate is below 5 deg/s after 2 complete orbits.                                                                             |
| 7.2.4     | Launcher Separation Rate   | The ADCS shall be able to detumble the spacecraft given the worst case launcher separation rate conditions of 100 deg/s.                                                                       |
| 7.2.5     | ADCS Coverage              | The ADCS shall provide the required mission pointing performance for at least 10 consecutive orbits, at a time.                                                                                |
| 7.2.6     | Wheel off-loading          | The ADCS shall ensure continuous wheel off-loading, not requiring any mission interruption for this purpose, while also keeping the stored angular momentum below 75% of the rated maximum.    |
| 7.2.7     | Average power consumption  | The ADCS shall ensure a average power consumption below 0.4 Watts.                                                                                                                             |

Table B.2: ADCS Operational Requirements.

| <b>Id</b> | <b>Performance Req.</b>           | <b>Requirement Description</b>                                                                                                                                                                            |
|-----------|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7.3.1     | Absolute attitude normal pointing | The ADCS shall ensure during the operational mission phase an absolute pointing performance of 1 degree, at 95% confidence level, in normal mode (temporal statistical interpretation).                   |
| 7.3.2     | Absolute attitude safe pointing   | The ADCS shall ensure during the operational mission phase an absolute pointing performance of 10 degree, at 95% confidence level, in safe mode and out of eclipse (temporal statistical interpretation). |
| 7.3.3     | Absolute attitude knowledge       | The ADCS shall ensure during the operational mission phase an on-board absolute attitude knowledge performance of 2.5 degree, at 95% confidence level (temporal statistical interpretation).              |
| 7.3.4     | Orbit Knowledge                   | The ADCS shall ensure during the operational mission phase an on-board orbit estimation with an accuracy of 250 meters, in the ECI frame, at 95% confidence level (temporal statistical interpretation).  |

Table B.3: ADCS Performance Requirements.

| <b>Id</b> | <b>Design Const.</b>           | <b>Constraint Description</b>                                                                                                                                                                                                                                   |
|-----------|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7.4.1     | OBC Interaction                | The ADCS software shall be executed in the OBC hardware.                                                                                                                                                                                                        |
| 7.4.2     | Control Scheme                 | The ADCS shall use active control techniques.                                                                                                                                                                                                                   |
| 7.4.3     | Attitude Determination         | The ADCS shall include on-board attitude filtering.                                                                                                                                                                                                             |
| 7.4.4     | ADCS Design Space              | The ADCS shall include a gyroscope, a magnetometer, one CSS in each face (except Z-), and three magnetic torquers mounted orthogonally. It may only additionally include an earth sensor mounted in the face Z- and three reaction-wheels mounted orthogonally. |
| 7.4.5     | Spacecraft Position Estimation | The ADCS shall include on-board estimation of the spacecraft position represented in the ECI frame.                                                                                                                                                             |

Table B.4: ADCS Design Constraints.

## Appendix C

# Spacecraft Environment and Orbital Dynamics

### C.1 Expansion in Spherical Harmonics

The determination of the Earth's local magnetic field vector and acceleration of gravity relies on the expansion in spherical harmonics. This expansion uses a term  $P_{n,m}$  called the associated Legendre polynomial (of degree  $n$  and order  $m$ ). For any real number  $\mu$ , this term is defined as [95]

$$P_{n,m}(\mu) = \frac{1}{2^n n!} (1 - \mu^2)^{m/2} \frac{d^{n+m}}{d\mu^{n+m}} (\mu^2 - 1)^n. \quad (C.1)$$

The associated Legendre polynomial can be calculated recursively [95]. Starting with  $P_{0,0} = 1$ , all the polynomials  $P_{m,m}$  are computed with

$$P_{m,m}(\mu) = (2m - 1) \sqrt{\mu^2 - 1} P_{m-1,m-1}(\mu). \quad (C.2)$$

Using the values defined with the previous equation, the polynomials  $P_{m+1,m}$  are computed with

$$P_{m+1,m}(\mu) = (2m + 1) \mu P_{m,m}(\mu), \quad (C.3)$$

and the rest of the polynomials, for  $n > m + 1$ , are given by

$$P_{n,m}(\mu) = \frac{1}{n - m} ((2n - 1) \mu P_{n-1,m}(\mu) - (n + m - 1) P_{n-2,m}(\mu)). \quad (C.4)$$

For any real number  $\mu$ ,  $\check{P}_{n,m}(\mu)$  are the Schmidt semi-normalized associated Legendre functions, which are defined as

$$\begin{aligned} \check{P}_{n,m}(\mu) &= \sqrt{2 \frac{(n-m)!}{(n+m)!}} P_{n,m}(\mu) \quad \text{if } m > 0, \\ \check{P}_{n,m}(\mu) &= P_{n,m}(\mu) \quad \text{if } m = 0. \end{aligned} \quad (C.5)$$

## C.2 Earth Magnetic Field

The Earth magnetic field was modelled using the World Magnetic Model (WMM) 2020, which is described in [92]. The magnetic field is a potential field such that the magnetic field vector  $\mathbf{B}$  can be written as the negative spatial gradient of a scalar potential. Using geocentric spherical coordinates we define the magnetic field in the NED frame  $\mathbf{B}_H = \{B_{H_x}, B_{H_y}, B_{H_z}\}$  as:

$$\mathbf{B}_H(\lambda, \phi', r, t) = -\nabla V(\lambda, \phi', r, t) = \left\{ -\frac{1}{r} \frac{\partial V}{\partial \phi'}, -\frac{1}{r \cos \phi'} \frac{\partial V}{\partial \lambda}, \frac{\partial V}{\partial r} \right\}, \quad (\text{C.6})$$

where  $t$  represents time. The scalar potential  $V$  can be expanded in terms of spherical harmonics as [92]

$$V(\lambda, \phi', r, t) = R_{\oplus} \sum_{n=1}^N \left( \frac{R_{\oplus}}{r} \right)^{n+1} \sum_{m=0}^n (g_{n,m}(t) \cos(m\lambda) + h_{n,m}(t) \sin(m\lambda)) \check{P}_{n,m}(\sin \phi'), \quad (\text{C.7})$$

where  $N$  is the degree of the expansion of the WMM, and  $g_{n,m}(t)$  and  $h_{n,m}(t)$  are the time-dependent Gauss coefficients of degree  $n$  and order  $m$  describing the Earth's main magnetic field (values for these coefficients are specified in [92] for  $n, m \leq 12$ ). These coefficients vary linearly with time, and in this work we use the coefficient values correspondent to the instant of launcher separation. The degree of the expansion  $N$  is a design parameter of the magnetic field vector estimator, which introduces a trade-off between the on-board computational effort and memory usage, and the accuracy of the estimation. The components of the magnetic field vector along each axis of the NED frame are given by

$$\begin{aligned} B_{H_x} &= -\frac{1}{r} \frac{\partial V}{\partial \phi'} = -\sum_{n=1}^N \left( \frac{R_{\oplus}}{r} \right)^{n+2} \sum_{m=0}^n [g_{n,m}(t) \cos(m\lambda) + h_{n,m}(t) \sin(m\lambda)] \frac{d\check{P}_{n,m}(\sin \phi')}{d\phi'}, \\ B_{H_y} &= -\frac{1}{r \cos \phi'} \frac{\partial V}{\partial \lambda} = \frac{1}{\cos \phi'} \sum_{n=1}^N \left( \frac{R_{\oplus}}{r} \right)^{n+2} \sum_{m=0}^n m [g_{n,m}(t) \sin(m\lambda) - h_{n,m}(t) \cos(m\lambda)] \check{P}_{n,m}(\sin \phi'), \\ B_{H_z} &= \frac{\partial V}{\partial r} = -\sum_{n=1}^N (n+1) \left( \frac{R_{\oplus}}{r} \right)^{n+2} \sum_{m=0}^n [g_{n,m}(t) \cos(m\lambda) + h_{n,m}(t) \sin(m\lambda)] \check{P}_{n,m}(\sin \phi'). \end{aligned} \quad (\text{C.8})$$

where  $d\check{P}_{n,m}(\sin \phi')/d\phi'$  is defined as

$$\frac{d\check{P}_{n,m}(\sin \phi')}{d\phi'} = (n+1)(\tan \phi') \check{P}_{n,m}(\sin \phi') - \sqrt{(n+1)^2 - m^2} (\sec \phi') \check{P}_{n+1,m}(\sin \phi'). \quad (\text{C.9})$$

## C.3 Earth Gravity Field

The Earth's acceleration of gravity  $\mathbf{g}$  can also be computed as the spatial gradient of a scalar potential, the gravitational potential  $U$ . The acceleration of gravity in the NED frame  $\mathbf{g}_H = \{g_{H_x}, g_{H_y}, g_{H_z}\}$  is given by

$$\mathbf{g}_H(\lambda, \phi', r, t) = \nabla U(\lambda, \phi', r, t) = \left\{ \frac{1}{r} \frac{\partial U}{\partial \phi'}, \frac{1}{r \cos \phi'} \frac{\partial U}{\partial \lambda}, -\frac{\partial U}{\partial r} \right\}, \quad (\text{C.10})$$

where  $t$  represent time. If we consider that the total mass of Earth (or any other central body) is concentrated in the centre of the coordinate system, the gravitational potential is given by

$$U = \frac{\mu_{\oplus}}{r}. \quad (\text{C.11})$$

A more accurate estimate of the gravitational potential can be derived by applying an expansion in spherical harmonics, resulting in the equation [98]

$$U = \frac{\mu_{\oplus}}{r} \left( 1 + \sum_{n=2}^N \left( \frac{R_{\oplus}}{r} \right)^n \sum_{m=0}^n (C_{n,m} \cos(m\lambda) + S_{n,m} \sin(m\lambda)) \check{P}_{n,m}(\sin \phi') \right), \quad (\text{C.12})$$

where  $N$  is the degree of the expansion, and  $C_{n,m}(t)$  and  $S_{n,m}(t)$  are the geopotential coefficients of degree  $n$  and order  $m$  describing the Earth gravity field (values for these coefficients are specified in [102] for  $n, m \leq 2159$ ). The coefficients must be used in these equations with the same type of normalization as the Legendre polynomials. The degree of the expansion  $N$  is a design parameter of the spacecraft position estimator, which introduces a trade-off between the on-board computational effort and memory usage, and the accuracy of the estimation. The components of the acceleration of gravity along each axis of the NED frame are given by

$$\begin{aligned} g_{H_x} &= \frac{1}{r} \frac{\partial U}{\partial \phi'} = \frac{\mu_{\oplus}}{r^2} \sum_{n=2}^N \left( \frac{R_{\oplus}}{r} \right)^n \sum_{m=0}^n [C_{n,m} \cos(m\lambda) + S_{n,m} \sin(m\lambda)] [\check{P}_{n,m+1}(\sin \phi') - m \tan(\lambda) \check{P}_{n,m}(\sin \phi')], \\ g_{H_y} &= \frac{1}{r \cos \phi'} \frac{\partial U}{\partial \lambda} = \frac{\mu_{\oplus}}{\cos \phi' r^2} \sum_{n=2}^N \left( \frac{R_{\oplus}}{r} \right)^n \sum_{m=0}^n m [S_{n,m} \cos(m\lambda) - C_{n,m} \sin(m\lambda)] \check{P}_{n,m}(\sin \phi'), \\ g_{H_z} &= -\frac{\partial U}{\partial r} = -\frac{\mu_{\oplus}}{r^2} \sum_{n=2}^N \left( \frac{R_{\oplus}}{r} \right)^n (n+1) \sum_{m=0}^n [C_{n,m} \cos(m\lambda) + S_{n,m} \sin(m\lambda)] \check{P}_{n,m}(\sin \phi'). \end{aligned} \quad (\text{C.13})$$

## C.4 Earth Albedo

The model of the Earth albedo developed in [93] has been used in this work with some modifications. The major modification concerns the division of the albedo computation into two parts, one that does not depend on the knowledge of the spacecraft attitude and another that does. The benefit of this approach is that the part of the equations that do not depend on the attitude can be executed at a slower frequency, saving processing power. This model is based on reflectivity data measured by the Earth Probe Satellite for the TOMS project (data available in [103]). The data used in this work was the average of the years of 2001 through 2005. The ECEF frame is the only frame used in the formulation of this model, and so the subscript  $E$  is omitted to facilitate the understanding of the equations. The surface of the Earth is divided into a grid of size  $N \times M$  composed of data points  $\Phi \times \Lambda$ , where

$$\begin{aligned} \Phi &= [\Delta\phi_g/2, 3\Delta\phi_g/2, 5\Delta\phi_g/2, \dots, \pi - \Delta\phi_g/2], \\ \Lambda &= [\Delta\lambda_g/2, 3\Delta\lambda_g/2, 5\Delta\lambda_g/2, \dots, 2\pi - \Delta\lambda_g/2]. \end{aligned} \quad (\text{C.14})$$

Each data point is denoted here as  $(\phi_g^i, \lambda_g^j)$  and the grid resolution used is such that  $\Delta\phi_g = 2 \text{ deg}$  and  $\Delta\lambda_g = 2.5 \text{ deg}$ . This model considers that the solar irradiance  $E_{\text{AM0}}$  reaches each cell at the data point  $(\phi_g^i, \lambda_g^j)$ , and at an incident angle of  $\alpha_{\text{Sun}}^{i,j}$  with the outward normal of the cell in question  $\mathbf{n}_c^{i,j}$ . The area of the grid cells is given by

$$S_c(\phi_g^i, \lambda_g^j) = \lambda_g^j R_\oplus \left( \cos \left( \phi_g^i - \frac{\Delta\phi_g}{2} \right) - \cos \left( \phi_g^i + \frac{\Delta\phi_g}{2} \right) \right), \quad (\text{C.15})$$

and the solar irradiance at Earth is defined as  $E_{\text{AM0}} = 1366.5 \text{ W m}^{-2}$  (refer to [93] for a mathematical demonstration). The incident radiant flux on a single cell is then given by

$$P_c(\phi_g^i, \lambda_g^j) = E_{\text{AM0}} S_c(\phi_g^i, \lambda_g^j) \left\{ \mathbf{n}_c^{i,j} \cdot \mathbf{r}_{\text{Sun}} / \|\mathbf{r}_{\text{Sun}}\| \right\}_0^\infty, \quad (\text{C.16})$$

where the notation  $\{\cdot\}_0^\infty$  denotes a lower saturation of zero, so that the incident radiant flux is set to zero for all the cells that are not illuminated by the Sun. Assuming Lambertian reflectivity, the irradiance from a single cell at the satellite is given by

$$E_c(\phi_g^i, \lambda_g^j) = \frac{\rho_c(\phi_g^i, \lambda_g^j) P_c(\phi_g^i, \lambda_g^j) \left\{ \mathbf{n}_c^{i,j} \cdot \mathbf{r}_{\text{sat}} / \|\mathbf{r}_{\text{sat}}\| \right\}_0^\infty}{\pi \|\mathbf{r}_{\text{sat}}\|^2}, \quad (\text{C.17})$$

where  $\rho_c(\phi_g^i, \lambda_g^j)$  denotes the reflectivity of the cell in question. This equation accounts for the dependence of the irradiance on the visible area of the cell surface seen from the spacecraft location. The saturation in this equation reflects that a cell that is not visible from the spacecraft cannot contribute to the albedo irradiance. Up to the calculation of  $E_c(\phi_g^i, \lambda_g^j)$  for each cell, the spacecraft attitude is not used. This part of the albedo model is represented in this work by the block named *Earth Albedo* (see Section 4.2.6).

The second part of the albedo model consists of the calculation of the total irradiance at each of the CSSs. The irradiance of the Sun is added to the irradiance of the Earth albedo, each at a specific incident angle. Denoting the normal of a specific face  $k$  of the spacecraft as  $\mathbf{n}_s^k$ , the total irradiance at each CSS is given by

$$E^k = \sum_i^N \sum_j^M \left( E_c(\phi_g^i, \lambda_g^j) \left\{ \mathbf{n}_s^k \cdot \frac{\mathbf{r}_c^{i,j} - \mathbf{r}_{\text{sat}}}{\|\mathbf{r}_c^{i,j} - \mathbf{r}_{\text{sat}}\|} \right\}_0^\infty \right) + E_{\text{AM0}} \left\{ \mathbf{n}_s^k \cdot \frac{\mathbf{r}_{\text{Sun}} - \mathbf{r}_{\text{sat}}}{\|\mathbf{r}_{\text{Sun}} - \mathbf{r}_{\text{sat}}\|} \right\}_0^\infty \quad (\text{C.18})$$

where  $\mathbf{r}_c^{i,j}$  denotes the position of the data point  $(\phi_g^i, \lambda_g^j)$  in the ECEF frame. This part of the albedo model is represented in this work by the block named *Photodiodes Interface* (see Section 4.2.6).

## Appendix D

# Mathematical Models of Components

### D.1 Hardware

#### D.1.1 Sensors

In this section the sensor measurement models are presented. Discretization of the signals with a defined frequency, linearity errors, and saturations on the final measured values are added to these models, when applicable, resulting in the final mathematical model.

**Gyroscope** The measurement model used for the gyroscope is given by [104]

$$\hat{\omega} = \omega + b_g + \mu_{\text{arw}}, \quad (\text{D.1})$$

$$\dot{b}_g = \mu_{\text{rrw}}, \quad (\text{D.2})$$

where  $\hat{\omega}$  is the measured angular velocity,  $\omega$  is the true angular velocity,  $b_g$  is the gyro drift bias driven by the RRW process  $\mu_{\text{rrw}}$ , and  $\mu_{\text{arw}}$  is white noise that corresponds to an ARW process. The processes  $\mu_{\text{arw}}$  and  $\mu_{\text{rrw}}$  are both defined with power spectral densities (PSD), which are defined in Table 4.3.

**Magnetometer** The measurement model used for the magnetometer is given by

$$\hat{B}_B = B_B + b_m + \mu_m, \quad (\text{D.3})$$

where  $b_m$  represents a bias and  $\mu_m$  represents a noise component. This noise is modelled as Gaussian white noise with a variance which is defined in Table 4.3.

**Photodiode** The model used for the photodiode is ideal, as no information was available in the datasheet to support the selection of a different model. For each photodiode  $k$ , its current output  $i_k$  is given by

$$i_k = k_{\text{pd}} E_k, \quad (\text{D.4})$$

where  $k_{pd}$  is a conversion factor, and  $E_k$  is the total irradiance defined in Eq. C.18.

**Earth Sensor** The measurement model used for the Earth sensor is given by

$$\hat{\mathbf{n}}_B = \mathbf{n}_B \cos(\epsilon_{es}) + \frac{\mathbf{n}_B \times \mathbf{e}_{B_x}}{\|\mathbf{n}_B \times \mathbf{e}_{B_x}\|} \sin(\epsilon_{es}), \quad (\text{D.5})$$

where  $\mathbf{n}_B$  is the nadir in the body frame, ( $\mathbf{e}_{B_x} = \{1, 0, 0\}$ ), and  $\epsilon_{es}$  is an angular error noise process, with a standard deviation as defined in Table 4.3. For simplicity, the angular error is applied around  $\mathbf{e}_{B_x}$ , which does not have a significant influence on the attitude knowledge error since the rotation around the z axis of the body frame is not controlled. In the mathematical model of the Earth sensor, no reading of the nadir is available when the Earth is not fully contained inside its FOV, defined in Table 4.3.

## D.1.2 Actuators

**Magnetic Torquer** The OBC controls the magnetic torquers by providing a PWM signal with a specific dutycycle. The dutycycle of one magnetic torquer is defined as the ratio between the desired dipole and the maximum dipole that can be produced by the torquer. The mathematical model of this actuator does not include the PWM signal, for simplicity. The actuation produced by one torquer is computed as:

$$\mathbf{T}_{mtq} = m_{\max} D \mathbf{n}_{mtq} \times \mathbf{B}_B, \quad (\text{D.6})$$

where  $m_{\max}$  is the maximum dipole (defined in Table 4.4),  $D \leq D_{\max}$  is the dutycycle computed by the controllers ( $D_{\max}$  is defined in Table 4.4), and  $\mathbf{n}_{mtq}$  is the direction of the magnetic dipole produced.

**Reaction Wheel** An ideal model is used for the reaction wheels, for simplicity. For each wheel

$$\mathbf{T}_w = \dot{h}_w \mathbf{n}_w = I_w \dot{\omega}_w \mathbf{n}_w, \quad (\text{D.7})$$

where  $h_w$  is the angular momentum stored by the reaction wheel,  $\mathbf{n}_w$  represents the axis of rotation of the wheel,  $\omega_w$  is its angular speed (limited to the value defined in Table 4.4), and  $I_w$  is its moment of inertia (defined in Table 4.4). The maximum torque that can be produced by a reaction wheel is defined in Table 4.4, and it is implemented in this model as a saturation.

## D.2 Software

### D.2.1 Estimation Algorithms

#### Spacecraft and Earth Orbit Propagation

The numerical integration of the spacecraft and Earth orbits is based on the fourth-order Runge-Kutta (RK4) method. For a vector  $\mathbf{X} = [\mathbf{r} \ \mathbf{v}]^T = [x \ y \ z \ \dot{x} \ \dot{y} \ \dot{z}]^T$ , the numerical integration of the equations of

motion is designed to solve the equation given by

$$\dot{\mathbf{X}} = \mathbf{f}(t, \mathbf{X}). \quad (\text{D.8})$$

The formulation of the RK4 integration method is such that

$$\mathbf{X}_{i+1} = \mathbf{X}_i + h\left(\frac{1}{6}\hat{\mathbf{f}}_1 + \frac{1}{3}\hat{\mathbf{f}}_2 + \frac{1}{3}\hat{\mathbf{f}}_3 + \frac{1}{6}\hat{\mathbf{f}}_4\right), \quad (\text{D.9})$$

where  $h$  represents the step size of the integration, and  $\hat{\mathbf{f}}_1$ ,  $\hat{\mathbf{f}}_2$ ,  $\hat{\mathbf{f}}_3$ , and  $\hat{\mathbf{f}}_4$  are given by

$$\begin{aligned} \hat{\mathbf{f}}_1 &= \hat{\mathbf{f}}(t_i, \mathbf{X}_i), \\ \hat{\mathbf{f}}_2 &= \hat{\mathbf{f}}\left(t_i + \frac{1}{2}h, \mathbf{X}_i + \frac{1}{2}h\hat{\mathbf{f}}_1\right), \\ \hat{\mathbf{f}}_3 &= \hat{\mathbf{f}}\left(t_i + \frac{1}{2}h, \mathbf{X}_i + \frac{1}{2}h\hat{\mathbf{f}}_2\right), \\ \hat{\mathbf{f}}_4 &= \hat{\mathbf{f}}(t_i + h, \mathbf{X}_i + h\hat{\mathbf{f}}_3). \end{aligned} \quad (\text{D.10})$$

The estimate  $\hat{\mathbf{f}}(t_i, \mathbf{X}_i)$  is given by

$$\hat{\mathbf{f}}(t_i, \mathbf{X}_i) = [\hat{\mathbf{v}}_i \ \hat{\mathbf{a}}_i] = [(\mathbf{X}_i)_{4:6} \ \hat{\mathbf{a}}_i], \quad (\text{D.11})$$

where  $\hat{\mathbf{a}}_i$  is the estimate of the acceleration at the instant  $i$ . For the Earth orbit propagation, only the point mass gravity exerted by the Sun and Moon are considered for the estimation of this acceleration. For the spacecraft orbit propagation, only the Earth's acceleration of gravity is considered, and it is estimated using the expansion in spherical harmonics discussed in Section C.3 (the order of the expansion is defined in Table 4.5). All the other orbital perturbations are too small to affect the satisfaction of Req. 7.3.4. The nadir is estimated in the inertial frame as the normalized vector parallel and opposite to the spacecraft position vector estimate. In this work, we use the geocentric nadir in simulation for simplicity.

### Sun Vector Determination

The Sun vector estimate in the body frame is derived from the photodiode currents using the following equations:

$$\hat{\mathbf{u}}_{\text{Sun}} = \frac{1}{\sqrt{i_x^2 + i_y^2 + i_z^2}} \begin{bmatrix} i_x \\ i_y \\ i_z \end{bmatrix}, \quad (\text{D.12})$$

$$\begin{cases} i_x = i_{X+} & \text{if } i_{X+} > i_{X-}, \\ i_x = -i_{X-} & \text{if } i_{X+} \leq i_{X-}, \\ i_y = i_{Y+} & \text{if } i_{Y+} > i_{Y-}, \\ i_y = -i_{Y-} & \text{if } i_{Y+} \leq i_{Y-}, \\ i_z = i_{Z+}. \end{cases} \quad (\text{D.13})$$



where  $i_{X^+}$  represents the current of the photodiode installed in the face  $X^+$ , which is always positive or null (the same applies to all the other faces). The three currents  $i_x$ ,  $i_y$ , and  $i_z$  are intermediate variables used to derive a Sun vector estimate  $\hat{\mathbf{u}}_{\text{Sun}}$ .

## Attitude Filtering

The attitude filter used in this work is the Explicit Complementary Filter for attitude determination, which was developed in [105]. The formulation of this filter is expressed here in the body frame and in quaternion form (attitude representation and quaternion mathematics are discussed in Appendix A.2). We may define the true quaternion  $\mathbf{q}^{\text{true}}$  in terms of an estimated quaternion  $\hat{\mathbf{q}}$  as

$$\mathbf{q}^{\text{true}} = \tilde{\mathbf{q}} \otimes \hat{\mathbf{q}}, \quad (\text{D.14})$$

where  $\tilde{\mathbf{q}}$  is the error quaternion. The goal of the filter design is to find kinematics for  $\hat{\mathbf{q}}(t)$  such that  $\tilde{\mathbf{q}}(t) \rightarrow \mathbf{I}_q$ , where  $\mathbf{I}_q$  is the identity quaternion. Considering the gyroscope measurement model defined in Eq. D.1 and using the quaternion representation of the attitude kinematics described in Eq. A.23, the kinematics of  $\hat{\mathbf{q}}(t)$  can be defined as

$$\dot{\hat{\mathbf{q}}} = \frac{1}{2} \Xi(\hat{\mathbf{q}})(\hat{\boldsymbol{\omega}} + \boldsymbol{\beta}_g + k_p \boldsymbol{\gamma}), \quad (\text{D.15})$$

where  $\boldsymbol{\beta}_g$  is the estimate of the measurement bias,  $k_p$  is a positive scalar gain, and  $\boldsymbol{\gamma}$  is a correction factor based on the estimation of the current attitude error. The correction factor  $\boldsymbol{\gamma}$  is determined using the measurements of known vector quantities and is given by

$$\boldsymbol{\gamma} = \sum_i k_i (\mathbf{s}_B^i \times \hat{\mathbf{A}} \mathbf{s}_I^i), \quad (\text{D.16})$$

where  $\hat{\mathbf{A}} \equiv C_q(\hat{\mathbf{q}})$  is the estimated attitude matrix,  $k_i$  is a positive scalar gain relative to the  $i^{\text{th}}$  measured vector quantity,  $\mathbf{s}_B^i$  is the  $i^{\text{th}}$  vector measurement in the frame  $B$ , and  $\mathbf{s}_I^i$  is the estimation of the  $i^{\text{th}}$  vector quantity in the frame  $I$ .

A given vector measurement provides no attitude information regarding the rotation around that vector. For that reason, even with very accurate sensors, at least two measurements of known non-colinear vectors are required. In the case of two vector measurements with different accuracies being used, we can limit the effect of the less accurate measurement (sensor  $b$ ) to the rotation around the more accurate measurement (sensor  $a$ ) such that  $\boldsymbol{\gamma}$  is given by [106]

$$\boldsymbol{\gamma} = k_a (\mathbf{s}_B^a \times \hat{\mathbf{A}} \mathbf{s}_I^a) + k_b ((\mathbf{s}_B^b \times \hat{\mathbf{A}} \mathbf{s}_I^b) \cdot \mathbf{s}_B^a) \mathbf{s}_B^a. \quad (\text{D.17})$$

The estimation of the gyroscope bias  $\boldsymbol{\beta}_g$  is integrated at each step and the time derivative  $\dot{\boldsymbol{\beta}}_g$  is given by

$$\dot{\boldsymbol{\beta}}_g = -k_g \boldsymbol{\gamma}. \quad (\text{D.18})$$

Finally, a corrected estimate of the angular velocity may be derived with the equation

$$\hat{\omega}_{\text{corr}} = \hat{\omega} + \beta_g. \quad (\text{D.19})$$

## D.2.2 Control Algorithms

### Detumbling

In this work, a detumbling algorithm based on magnetic torquers is used. The algorithm and mathematical proof of global asymptotic convergence were developed in [107]. The commanded magnetic dipole moment is given by

$$\mathbf{m}_d = \frac{k_d}{\|\hat{\mathbf{B}}_B\|^2} \hat{\omega} \times \hat{\mathbf{B}}_B, \quad (\text{D.20})$$

where  $k_d$  is a positive scalar gain and  $\hat{\mathbf{B}}_B$  is the magnetic field vector measurement in the body frame. With this control law, assuming perfect sensing, the control torque is given by (where  $\mathbf{b} = \mathbf{B}_B/\|\mathbf{B}_B\|$ ):

$$\boldsymbol{\tau}_d = \mathbf{m}_d \times \mathbf{B}_B = \frac{k_d}{\|\mathbf{B}_B\|^2} (\boldsymbol{\omega} \times \mathbf{B}_B) \times \mathbf{B}_B = k_d (\boldsymbol{\omega} \times \mathbf{b}) \times \mathbf{b} = -k_d(I_{3 \times 3} - \mathbf{b}\mathbf{b}^T) \boldsymbol{\omega}. \quad (\text{D.21})$$

The control torque's direction is as aligned as possible with the direction opposite to the angular velocity vector, and its magnitude is proportional to the cosine of the angle between the former and latter directions. This algorithm uses the gyroscope measurements because these are considered to be reliable. Otherwise, the B-dot algorithm can be used, which is based on magnetometer measurements.

### Pointing

The algorithms used for pointing are based on PD control, which is also used in the ISTSat-1. The objective of these algorithms is to keep the outward normal of the face  $Z^-$  aligned with the nadir (face nomenclature is defined in Appendix A.1.4). Thus, the rotation about the z axis of the body frame is not completely stabilized. In this work, this type of algorithm is seen as a black-box and we assumed that it has been validated in terms of its performance and stability. The formulation for the basic PD controller is such that the desired control torque is defined as

$$\boldsymbol{\tau}_{\text{rw}} = k_\epsilon \boldsymbol{\epsilon} - k_\omega \hat{\omega}_{\text{corr}}, \quad (\text{D.22})$$

where  $k_\epsilon$  and  $k_\omega$  are positive scalar gains. The desired rotation vector  $\boldsymbol{\epsilon}$  is given by

$$\boldsymbol{\epsilon} = \mathbf{n}_{\text{pld}} \times \hat{\mathbf{n}}_B = \mathbf{n}_{\text{pld}} \times A \hat{\mathbf{n}}_I = A \frac{\hat{\mathbf{r}}_I}{\|\hat{\mathbf{r}}_I\|} \times \mathbf{n}_{\text{pld}}, \quad (\text{D.23})$$

where  $\mathbf{n}_{\text{pld}} = \{0, 0, -1\}$  denotes the orientation, in the body frame, that must be aligned with the local nadir. This basic PD control formulation is used for reaction wheel-based pointing, since this type of actuator can produce torques in any desired direction. For magnetic torquer-based pointing, a modification is made to the basic formulation, following the design of the ADCS of the ISTSat-1. During the

development of the ISTSat-1, simulations run for the basic PD control formulation showed that the control component of the pointing error was too high (the angular difference between the estimated nadir in the body frame and the actual direction of  $\mathbf{n}_{\text{pld}}$ ). This was due to the fact that magnetic torquers cannot produce torque in every direction. To solve this issue, an additional term was added to Eq. D.22, introducing a torque component that rotates the angular velocity of the spacecraft towards  $\epsilon$ . The desired torque is instead defined as

$$\boldsymbol{\tau}_{\text{mp,d}} = k_{\epsilon}\epsilon - k_{\omega}\hat{\boldsymbol{\omega}}_{\text{corr}} + k_r\epsilon_{\omega}, \quad (\text{D.24})$$

where  $k_r$  is a positive scalar gain. The angular velocity rotation vector  $\epsilon_{\omega}$  is given by

$$\epsilon_{\omega} = \left( \epsilon \times \frac{\hat{\boldsymbol{\omega}}_{xy}}{\|\hat{\boldsymbol{\omega}}_{xy}\|} \right) \times \frac{\hat{\boldsymbol{\omega}}_{xy}}{\|\hat{\boldsymbol{\omega}}_{xy}\|}, \quad (\text{D.25})$$

where  $\hat{\boldsymbol{\omega}}_{xy}$  is the projection of the angular velocity measurement onto the  $Oxy$  plane of the body frame, being given by

$$\hat{\boldsymbol{\omega}}_{xy} = \frac{\hat{\boldsymbol{\omega}}_{\text{corr}}}{\|\hat{\boldsymbol{\omega}}_{\text{corr}}\|} - \left( \frac{\hat{\boldsymbol{\omega}}_{\text{corr}}}{\|\hat{\boldsymbol{\omega}}_{\text{corr}}\|} \cdot \hat{\mathbf{n}}_B \right) \hat{\mathbf{n}}_B. \quad (\text{D.26})$$

The component of the desired control torque that is parallel to the local magnetic field vector cannot be produced. Also, the commanded magnetic dipole is defined to be orthogonal to the magnetic field vector, otherwise electric energy would be wasted. The derivation of the commanded magnetic dipole  $\mathbf{m}_{\text{mp}}$  from  $\boldsymbol{\tau}_{\text{mp,d}}$  is as follows:

$$\frac{\mathbf{B}_B \times \boldsymbol{\tau}_{\text{mp,d}}}{\|\mathbf{B}_B\|^2} = \frac{\mathbf{B}_B \times \boldsymbol{\tau}_{\text{mp}}}{\|\mathbf{B}_B\|^2} = \mathbf{b} \times (\mathbf{m}_{\text{mp}} \times \mathbf{b}) = (\mathbf{b} \cdot \mathbf{b})\mathbf{m}_{\text{mp}} - (\mathbf{b} \cdot \mathbf{m}_{\text{mp}})\mathbf{b} = \mathbf{m}_{\text{mp}}, \quad (\text{D.27})$$

where the actual control torque  $\boldsymbol{\tau}_{\text{mp}}$  is the projection of  $\boldsymbol{\tau}_{\text{mp,d}}$  onto a plane that is orthogonal to the magnetic field vector.

## Reaction Wheel Momentum Management

The desired torque to be used for reaction wheel off-loading is computed with a linear relation to the angular velocity of the reaction wheels, and it is given by

$$\boldsymbol{\tau}_{\text{mm,d}} = -k_{\text{mm}}\boldsymbol{\omega}_w \quad (\text{D.28})$$

where  $k_{\text{mm}}$  is a positive scalar gain and  $\boldsymbol{\omega}_w$  is the sum of the angular velocities of the three reaction wheels. The commanded magnetic dipole  $\mathbf{m}_{\text{mm}}$  is computed from  $\boldsymbol{\tau}_{\text{mm,d}}$  using the relation defined in Eq. D.27. An estimate of the actual torque produced is derived as

$$\boldsymbol{\tau}_{\text{mm}} = \mathbf{m}_{\text{mm}} \times \hat{\mathbf{B}}_B. \quad (\text{D.29})$$

A torque component  $-\boldsymbol{\tau}_{\text{mm}}$  is added to the reaction wheel control actuation defined in Eq. D.22.