Politecnico di Torino

Politecnico di Torino

Master's Degree in Aerospace Engineering
Academic Year 2024/2025
Graduation Session October 2025

Development of an ECSS-SMP-Compliant Satellite Digital Twin

Supervisor:
Prof. Paolo Maggiore

Tutor:
Ing. Carlo Maria Paccagnini

Candidate:
Lorenzo Bernardi

Abstract

The persistent fragmentation of software tools in spacecraft simulation has long hindered the reusability of high-value models, introducing programmatic risks for complex, multi-partner missions. To mitigate these issues, the European space industry has mandated compliance with the European Cooperation for Space Standardization Simulation Model Portability standard, promoting interoperability and long-term model sustainability.

This thesis applies this standards-driven methodology to the Italian Space Agency's In-Orbit Servicing mission, documenting the complete development and validation of the mission's Electric Power System Digital Twin. The model encompasses the Solar Array Wings, Solar Array Drive Assembly, Battery, and the central Power Control and Distribution Unit. Implemented in C++ using an object-oriented design, it accurately represents both the system's electrical behavior and its MIL-STD-1553 data bus interfaces. The Digital Twin was integrated into a dual-machine virtualized environment based on the ESA-reference SIMULUS 10.3.0 framework, replicating the operational separation between the spacecraft simulation and the ground control console.

Validation was conducted through comparative analysis against authoritative reference profiles from the official mission Power and Energy Budget document. In a nominal multi-orbit scenario (Case 1-2a), the predicted minimum bus voltage during eclipse stabilized at 53 V. In the more demanding high-power rendezvous scenario conducted entirely in eclipse (Case 6.2), the model accurately predicted a minimum voltage of approximately 52 V. Both results remained comfortably above the 50 V mission-success threshold, confirming the model's fidelity in the system's most critical, power-limited regimes.

This thesis therefore delivers a robust, validated, and fully SMP-compliant engineering asset, fit for its intended purpose as the foundational power source for the integration of future Digital Twin subsystem models. Beyond supporting upcoming mission-level verification activities, the work establishes a verifiable, standards-aligned framework that contributes a durable and reusable component to the broader European space simulation ecosystem.

Contents

List of Figures

List of Tables

Acronyms

AOCS Attitude and Orbit Control System

API Application Programming Interface

ARP Avionics and Robotics Platform

ASI Italian Space Agency

BCM Battery Charge Management

BOL Beginning-of-Life

BTA Battery

CC/CV Constant Current / Constant Voltage

CCSDS Consultative Committee for Space Data Systems

CNES French Space Agency

DOD Depth-of-Discharge

ECSS European Cooperation for Space Standardization

EOL End-of-Life

EOCV End-of-Charge Voltage

EPS Electric Power System

ESA European Space Agency

GNC Guidance, Navigation, and Control

GUI Graphical User Interface

HITL Human-in-the-Loop

HMI Human-Machine Interface

IDE Integrated Development Environment

IOS In-Orbit Servicing

ISIS Initiative for Space Innovative Standards

JPL Jet Propulsion Laboratory

JSF Joint Strike Fighter

LCL Latching Current Limiter

LEO Low Earth Orbit

MDE Model-Driven Engineering

NASA National Aeronautics and Space Administration

NEA Non-Explosive Actuator

OBC On-Board Computer

OOP Object-Oriented Programming

OSPM Orbital Propulsion and Support Module

PCDU Power Control and Distribution Unit

PDR Preliminary Design Review

PNRR National Recovery and Resilience Plan

PUS Packet Utilization Standard

RAII Resource Acquisition Is Initialization

S3R Sequential Switching Shunt Regulator

SADA Solar Array Drive Assembly

SADE Solar Array Drive Electronics

SADM Solar Array Drive Mechanism

SAW Solar Array Wing

SIMSAT Software Infrastructure for the Modeling of Satellites

SITL Software-in-the-Loop

SLES SUSE Linux Enterprise Server

SMP Simulation Model Portability

SOC State-of-Charge

SVE Software Validation Environment

TEMU Terma Emulator

TRL Technology Readiness Level

UMF Unified Modeling Framework

UML Unified Modeling Language

VM Virtual Machine

VOC Open-Circuit Voltage

1 Introduction

In aerospace engineering, a system simulator is a software platform used to model the dynamic interactions among subsystems and reproduce the behavior of an entire system. This virtual environment enables engineers to study complex phenomena, validate control strategies, and test operational scenarios that would otherwise be impractical, risky, or too costly to perform on physical hardware.

The effectiveness of a simulator depends on a fundamental trade-off between two key aspects: model fidelity, the accuracy with which physical reality is represented, and computational performance, the efficiency of execution and resource utilization. Managing this balance is a strategic consideration throughout the product lifecycle. During early design stages, engineers often rely on lower-fidelity models to rapidly explore different configurations. In later phases, however, high-fidelity simulations become essential for detailed verification and validation, helping identify design flaws and integration issues before committing to physical prototypes. By enabling early fault detection, simulation accelerates development, reduces technical and programmatic risks, and lowers overall project costs, making it a cornerstone of modern aerospace engineering.

Nevertheless, treating simulation purely as an offline tool for design and analysis is no longer sufficient for today's increasingly complex and autonomous space systems. These systems can evolve and face unforeseen conditions during operation, demanding continuous, real-time insight into their health and performance. Addressing this need has given rise to a new paradigm: the Digital Twin, which forms the central focus of this thesis.

1.1 The Digital Twin in the Space Domain

Unlike a traditional simulator, which is primarily used for design and pre-flight analysis, a Digital Twin in the space domain functions as a living model. It maintains a persistent, two-way data link with its physical counterpart, allowing it to continuously reflect the asset's actual operational state and evolve in real time.

The conceptual roots of this technology can be traced back to the Apollo 13 mission of the National Aeronautics and Space Administration (NASA). Following the critical oxygen tank explosion, ground engineers adapted the existing training

simulators by feeding them with the limited telemetry data received from the damaged spacecraft. This data-driven, ground-based replica effectively served as an early form of a Digital Twin, enabling mission control to test and validate recovery procedures within a virtual environment that closely mirrored the spacecraft's compromised condition. This pioneering approach demonstrated the immense value of coupling a physical asset with a synchronized virtual model to resolve complex operational challenges.

Building on this foundation, the Digital Twin has become an integral component of modern space missions, acting as a dynamic counterpart throughout a system's operational life. In system evolution, it provides a validated, high-fidelity environment for testing software updates and new operational procedures before deployment to the physical asset. In health management, it enables continuous state monitoring and early anomaly detection. It also plays a key role in mission safety and planning by simulating orbital environments, evaluating collision avoidance maneuvers, and providing a "safe-to-fail" environment for command validation. Furthermore, the Digital Twin supports performance optimization by allowing engineers to simulate and assess operational adjustments, such as changes in power consumption or thermal behavior, to extend mission life. Finally, it serves as a valuable training tool, offering an always up-to-date platform that helps operators remain proficient as the system evolves.

Collectively, these applications mark a significant shift from the traditional use of simulation as a pre-flight design and verification tool to its modern role as a continuous, data-driven resource for through-life operational support. As such, the Digital Twin represents a pivotal technology for enhancing the safety, longevity, and overall value of space assets. However, the absence of standardized methodologies for its development and integration remains a key challenge, that underscores the importance of adopting models aligned with established engineering standards.

1.2 The In-Orbit Servicing Project

The In-Orbit Servicing (IOS) project, led by the Italian Space Agency (ASI), is a strategic mission aimed at fostering a sustainable space ecosystem through the development of advanced in-orbit maintenance technologies. Its primary ob-

jective is to demonstrate autonomous capabilities across a range of critical servicing operations, including satellite repair and refueling, the assembly of large orbital structures, and the active removal of space debris. These capabilities are intended to address the growing issue of orbital congestion in Low Earth Orbit (LEO) and to extend the operational lifespan of high-value space assets.

Mission success depends on the integration of highly autonomous systems that leverage artificial intelligence and advanced Guidance, Navigation, and Control (GNC) technologies. Such systems must enable a servicing spacecraft to perform complex proximity operations, such as approaching and maneuvering around a target object, with exceptional precision and minimal risk, thereby reducing reliance on continuous Human-in-the-Loop (HITL) control. A key technological innovation within the mission is the development of a sophisticated robotic manipulator for capturing the target vehicle, as conceptually depicted in Figure 1. In the long term, these tools could revolutionize the construction of space infrastructure, making it possible to assemble large platforms incrementally in orbit rather than launching them as monolithic structures.

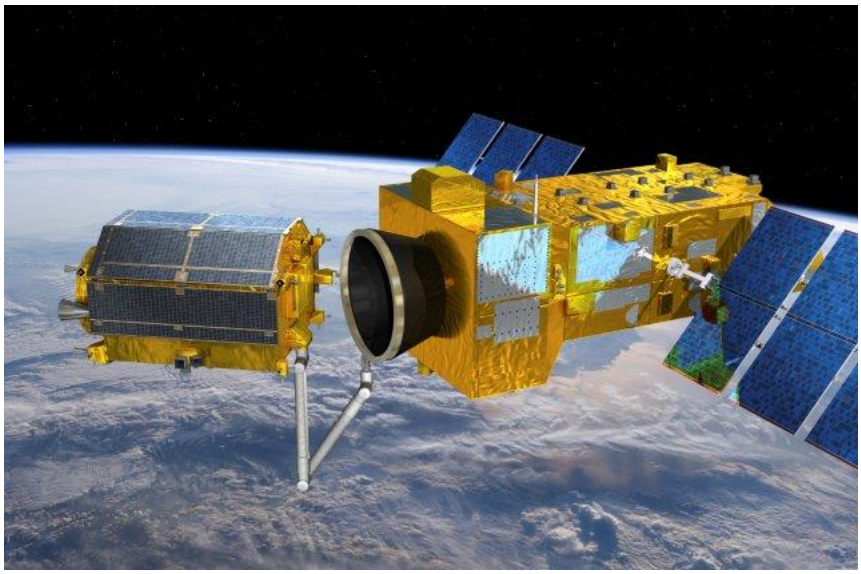

Figure 1: Render of the IOS Servicer performing a Capture Operation on a Target Satellite

To achieve these objectives, ASI has established a strong industrial partnership. Thales Alenia Space serves as the prime contractor, supported by Leonardo, Telespazio, Avio, and D-Orbit as key partners. Together, this con-

sortium brings extensive expertise in satellite systems, robotics, propulsion, and space operations, enabling the joint development and qualification of the IOS servicing spacecraft.

The mission is funded under Italy's National Recovery and Resilience Plan (PNRR), underscoring its national strategic significance. Scheduled for launch in 2026, the IOS mission aims to validate essential servicing technologies in a representative orbital environment. A core goal is to advance their Technology Readiness Level (TRL) from ground-tested maturity (TRL 5-6) to flight-proven status (TRL 7-8) through in-orbit demonstration. This milestone will lay the groundwork for future commercial satellite servicing activities and contribute to a more sustainable long-term human presence in space.

However, the combination of high autonomy, complex robotic operations, and the inherent risks of close-proximity maneuvers presents significant challenges for ground-based verification, validation, and operations. Conventional ground test campaigns cannot fully replicate the dynamic conditions of in-orbit interaction. This limitation highlights the critical need for a high-fidelity Digital Twin, a vital tool for mission simulation, operator training, real-time monitoring, and decision support. The development of this Digital Twin forms the central focus of this thesis.

1.3 Scope and Objectives of the Thesis

This thesis, conducted in collaboration with Thales Alenia Space in Turin, documents the development and validation of the first functional version of the Digital Twin for the IOS mission. The scope of this work extends beyond subsystem modeling to include the setup and configuration of the virtualized simulation environment required to host, execute, and validate the developed models. The final simulation environment is intended for delivery to Telespazio, the organization responsible for mission control, for direct use in upcoming mission-level activities.

A fundamental requirement guiding this work is strict compliance with the European Cooperation for Space Standardization (ECSS) Simulation Model Portability (SMP) standard. Adhering to this standard ensures that all developed models are modular, reusable, and interoperable, transforming them from project-specific implementations into long-term assets that can support future develop-

ments within the wider European space industry.

Within this framework, and in alignment with the principles of the SMP standard, the main objectives of this thesis are as follows:

1. Configuration of the simulation environment: To set up a virtualized simulation environment based on the SIMULUS framework, capable of hosting and executing SMP-compliant models.

2. Design and implementation of the Electric Power System (EPS) model: To develop a functional, SMP-compliant model of the satellite's EPS. The model focuses on simulating the electrical behavior of the core EPS components and their communication interfaces over the MIL-STD-1553 data bus. The EPS model is composed of the following interconnected sub-models:

    - Solar Array Wing (SAW): Simulates electrical power generation as a function of sun exposure and orbital position.
    - Solar Array Drive Assembly (SADA): Controls the orientation of the SAW to maximize solar incidence, based on telecommands received from the On-Board Computer (OBC) via the MIL-STD-1553 data bus.
    - Battery (BTA): Manages energy storage and provides power during eclipse periods or prior to SAW deployment.
    - Power Control and Distribution Unit (PCDU): Regulates power distribution and switches electrical loads in response to telecommands received from the OBC via the MIL-STD-1553 data bus.

3. Validation of the EPS Digital Twin: To define and execute a comprehensive set of test procedures that verify the EPS Digital Twin's behavior against expected operational scenarios.

To maintain a focused and achievable scope, other satellite subsystems, such as the Attitude and Orbit Control System (AOCS), communications, and propulsion, are excluded from this work. Likewise, detailed physical modeling beyond the electrical and communication behavior described above (for instance, thermal analysis) is intentionally omitted.

2 Foundations and State of the Art

2.1 ECSS-SMP Standard

The ECSS is a collaborative initiative involving the European Space Agency (ESA), national space agencies, and European industry. Its mission is to establish a coherent and harmonized set of standards that enhance the efficiency, consistency, and competitiveness of the European space sector. Within this framework, the SMP standard, formalized as ECSS-E-ST-40-07, was introduced to address a longstanding challenge of software fragmentation and limited model reusability in space system simulation.

Historically, the European simulation landscape was highly fragmented. Simulation models, often representing substantial investments in development and validation, were typically tied to proprietary software environments. This vendor lock-in severely limited interoperability between tools and organizations. As a result, models could not easily be reused or exchanged among different partners, even within the same mission. For large, collaborative programs, this inefficiency led to higher development costs, slower progress, and considerable duplication of effort. Moreover, it made the reuse of validated models across mission phases, such as transitioning from a software validation facility to an operational simulator, virtually impossible.

The SMP standard directly addresses these issues by defining a platform-independent architecture that separates the simulation model from the simulation environment. It does so by introducing a formal contract in the form of standardized software interfaces. Under this standard, each model is treated as a self-contained software component that must comply with a predefined communication protocol. This contract specifies how a model exposes its data and functionality to the outside world, distinguishing three key interaction types:

- Properties: Configurable parameters that act as “adjustment knobs” for the model (e.g., the maximum capacity of a battery).
- Operations: Commands that modify the model’s behavior, functioning like “control buttons” (e.g., a switchOn() command).
- Telemetry: Data outputs that report the model’s internal state, serving as

"dashboard gauges" (e.g., the current voltage output).

In addition to defining this standardized interface, SMP prescribes a well-structured component lifecycle managed by the simulator. During the Publish phase, the model declares all of its properties, operations, and telemetry elements. In the Configure phase, the simulator assigns initial values to these properties, typically sourced from configuration files. The Connect phase then links model interfaces, enabling telemetry from one model to drive property inputs in another. Finally, in the Execute phase, the simulator enters the main runtime loop, periodically invoking each model's update functions to advance its state in time.

To coordinate the integration of multiple standardized components, SMP employs a catalog-based system. Each model is described through XML files that specify its metadata, including its properties, interfaces, and dependencies. A higher-level assembly file defines how these individual components, potentially developed by different suppliers, are interconnected to form a complete spacecraft simulation.

By enforcing a standardized interface, data model, and lifecycle, the SMP standard creates an open and interoperable simulation ecosystem. It transforms models from isolated, project-specific code into reusable and portable assets that can be shared across organizations and missions. The effectiveness of this approach has been demonstrated through successful model exchanges between major European simulation frameworks, such as SIMULUS and SimTG. For the IOS mission, compliance with the SMP standard ensures that the Digital Twin models developed in this thesis are not merely mission-specific prototypes, but durable, reusable assets that contribute to the broader European space engineering community.

2.2 Space Simulation Frameworks

The development and deployment of a standardized simulation model, as mandated by ECSS-SMP, depend on a robust ecosystem of software frameworks. This section reviews the two principal environments that define the lifecycle of the model developed in this thesis. The first, SIMULUS, serves as the development framework in which the model is created, integrated, and validated. The second, the Initiative for Space Innovative Standards (ISIS) developed by the French

Space Agency (CNES), represents the operational ground segment framework and the ultimate deployment target for the model. Understanding the distinct roles and complementary relationship of these frameworks provides essential context for the engineering work presented herein.

2.2.1 SIMULUS: The Development and Validation Environment

SIMULUS is a reference framework developed by ESA for building high-fidelity, real-time operational simulators for European space missions. It functions not only as a simulation runtime, but as a comprehensive Model-Driven Engineering (MDE) ecosystem grounded in rigorous software engineering principles. The version employed in this thesis is 10.3.0.

At the core of SIMULUS lies the Unified Modeling Framework (UMF), which implements the MDE approach. In this paradigm, a formal visual design created in a compatible modeling tool serves as the master blueprint of the model. From this single design, the UMF toolchain automatically generates the necessary project artifacts, including the foundational software code structure, the ECSS-SMP XML catalogs, and the initial build system files. This workflow ensures that all components of the simulation remain synchronized with the central model definition throughout development.

Once these artifacts are generated, Apache Maven is used for build automation and dependency management. The compiled, SMP-compliant models are then executed within SIMSAT (Software Infrastructure for the Modeling of Satellites), the core runtime engine of the SIMULUS platform. SIMSAT's kernel is a service-oriented engine that provides essential SMP services such as time management, event scheduling, and synchronization, creating the live environment in which the Digital Twin operates.

Interaction with the running SIMSAT kernel can occur through several interfaces. A graphical Human-Machine Interface (HMI) enables direct monitoring and control, while a Scripting Service supports the automated assembly and execution of simulations. Most importantly, a Remote Connector exposes the simulation over a TCP/IP network interface, allowing external ground control software to connect to and interact with the live model. This network link forms a key element of the Digital Twin architecture, enabling real-time data exchange between the simulated and operational environments.

To ensure the integrity of this workflow, SIMULUS provides an integrated verification and validation framework supporting a structured testing hierarchy encompassing unit, integration, and system tests. The highest level of this process is Software-in-the-Loop (SITL) validation, in which the actual flight software is executed on an emulated space-grade processor and interacts directly with the simulated hardware models.

Within the scope of this thesis, this model-driven and validation-oriented environment provides the foundation for developing the EPS model, not merely as a code implementation, but as a robust, well-documented, and verifiable engineering asset fully compliant with the ECSS-SMP standard.

2.2.2 CNES ISIS: The Operational Target Environment

While SIMULUS serves as the development platform, the ISIS framework, developed by CNES, functions as a mission control system. Unlike SIMULUS, ISIS is not a simulator, but a complete ground segment infrastructure designed for satellite command, control, and monitoring. It provides all core control center services, telemetry acquisition and processing, telecommand management, and data archiving, built upon modern standards such as ECSS and the Consultative Committee for Space Data Systems (CCSDS).

Of particular relevance to this thesis is ISIS's ability to integrate and interact with SMP-compliant models. This interoperability enables a validated model developed in SIMULUS to be seamlessly integrated into the operational ground segment. Within this environment, the model can be driven by live spacecraft telemetry for predictive analysis, or used offline by operators to validate new procedures in a safe, controlled setting before their deployment to the actual spacecraft.

2.2.3 Model Lifecycle: From Validation to Operations

The relationship between SIMULUS and ISIS defines the two-stage lifecycle of the Digital Twin model. The ECSS-SMP standard serves as the critical bridge ensuring smooth transition between these stages. Initially, the model is developed, integrated, and validated within the high-fidelity engineering environment of SIMULUS. Once its behavior and performance are verified, the resulting stan-

dardized model, self-contained and fully compliant, is delivered as a portable asset ready for integration into the ISIS operational framework without modification.

This workflow transforms the model from a development artifact into a fully operational asset. Accordingly, the work presented in this thesis focuses on the first and most critical stage of this lifecycle: the development and validation of a robust, SMP-compliant model within the SIMULUS environment, engineered from the outset for eventual deployment in the operational ecosystem provided by ISIS.

2.3 Object-Oriented Programming

Object-Oriented Programming (OOP) serves as the foundational software paradigm for this thesis, providing a logical and scalable framework for modeling complex physical systems such as satellites. The core idea behind OOP is to structure software in a manner that reflects the physical architecture of the hardware, comprising discrete, interacting components. In this paradigm, each satellite subsystem, such as a battery, solar array, or actuator, is represented as an object: a self-contained entity that encapsulates both data (attributes) and functionality (methods).

OOP is built upon four key principles, encapsulation, inheritance, polymorphism, and abstraction, each of which plays a central role in the development of the IOS Digital Twin.

- Encapsulation involves grouping an object's data and methods together, effectively treating it as a "black box." For example, the SAW model encapsulates the internal logic governing power generation. Other subsystems, such as the PCDU, do not need to understand this internal logic; they simply interact with the SAW through its public interface using well-defined methods like setVoltage(v) or getCurrent(). This approach hides internal complexity, reduces coupling between components, and prevents unintended interactions within the system.

- Inheritance allows for the creation of a class hierarchy that promotes code reuse and structural consistency. A base class, such as Object, can define common attributes (e.g., name, description, parent) and generic methods shared across all subsystem models. Specific components. such as SAW,

BTA, or PCDU, then inherit from this base class, extending it with their own specialized behaviors. This mechanism minimizes code redundancy and ensures a coherent architectural structure across all models.

- Polymorphism, meaning “many forms,” enables different objects to be accessed through a shared interface, with each object responding according to its own implementation. For instance, the simulation scheduler can iterate through a collection of subsystem models and call a common method, such as updateState(deltaTime), on each one. The outcome varies by object: the BTA model updates its state-of-charge, while the SAW model computes its power output. This capability simplifies the simulation’s control logic and enhances extensibility.

- Abstraction focuses on managing complexity by exposing only the essential aspects of an object while concealing its internal details. For example, the SADA model does not need to reveal the detailed mechanics of its stepper motor. Instead, this complexity is encapsulated behind a high-level method such as rotateToAngle(angle). This abstraction makes the model easier to use and allows its internal fidelity to be increased later without affecting other parts of the simulation.

By adhering to these OOP principles, the software architecture of the Digital Twin mirrors the modular structure of the physical satellite. This approach yields a system that is intuitive to design, straightforward to maintain, and highly scalable. New components can be integrated into the simulation by simply implementing new classes that conform to established interfaces, often requiring minimal or no modification to the existing codebase.

2.4 C++ Programming Language

The selection of C++ as the implementation language for the IOS Digital Twin is a deliberate engineering choice informed by stringent technical requirements, such as computational performance and resource determinism, as well as essential programmatic constraints, including standards compliance and ecosystem compatibility.

High-performance execution is a fundamental requirement for a Digital Twin that must process real-time telemetry without introducing latency. As a compiled language, C++ offers low-level control over memory and processor resources while enabling aggressive compiler optimizations. This results in execution speeds that far exceed those of interpreted languages such as Python. Such performance is critical to maintaining synchronization between the simulation and the live data streams received from the spacecraft. A notable precedent is the CADAC++ architecture developed by the U.S. Air Force Research Laboratory, which demonstrates how object-oriented C++ simulations can effectively manage complex, multi-vehicle environments in real time.

Equally vital in mission-critical applications is deterministic control over system resources. Unlike languages that employ automatic garbage collection (e.g., Java), C++ provides developers with explicit control over memory allocation and deallocation. This eliminates the risk of unpredictable pauses caused by background memory management, an unacceptable behavior during time-sensitive operations such as satellite communication passes. The language's support for the Resource Acquisition Is Initialization (RAII) paradigm further enforces deterministic resource handling, ensuring that the simulation operates with consistent timing and reliability.

Compliance with industry standards also directly necessitates the use of C++. The ECSS-SMP standard, which underpins this work, defines its model-to-simulator interfaces explicitly in C++ through the use of abstract base classes. Implementing the models in C++ is therefore essential to achieve full compliance. This decision aligns the project with established best practices across major space organizations, including ESA and NASA, where C++ remains the dominant language for simulation and flight software. The robustness of C++ in safety-critical applications is further reflected in the existence of formalized coding standards, such as the Joint Strike Fighter (JSF) Air Vehicle C++ Coding Standard, developed to ensure software reliability and maintainability in aerospace systems.

Finally, C++ is indispensable for ecosystem compatibility. Both the SIMULUS simulation platform and the ISIS operational framework, core to this project, are natively implemented in C++ and expose their primary Application Programming Interfaces (APIs) through the language. Developing the Digital Twin in C++ ensures seamless integration with these frameworks and guarantees long-term

maintainability. Moreover, C++ provides access to a mature ecosystem of scientific and engineering libraries, such as Eigen for linear algebra and NASA's SPICE toolkit for astrodynamics. These tools enable the efficient implementation of complex physical and orbital models, further enhancing the fidelity of the simulation.

In summary, the adoption of C++ is not a matter of preference, but a direct response to the demanding performance, reliability, and integration requirements of the IOS Digital Twin. Its selection ensures that the developed models meet both the technical and programmatic standards necessary for their role within the broader European space simulation ecosystem.

2.5 State of the art in Aerospace Digital Twins

To properly contextualize the work presented in this thesis, it is essential to review the evolution and current state of Digital Twin technology within the aerospace sector. This overview traces the transition from static simulators to dynamic, data-driven models and highlights the key technological trends and applications that define the modern Digital Twin paradigm.

Although the practice of "twinning" can be traced back to NASA's Apollo program, as discussed in Section 1.1, the formal concept was first articulated by Dr. Michael Grieves in 2002 and later refined by NASA in 2011. This contemporary definition describes a multi-physics, multi-scale digital replica of an "as-built" physical asset that continuously ingests sensor data to mirror, predict, and optimize its operational life. This represented a fundamental shift from traditional simulators, transforming them into living models that evolve in parallel with their physical counterparts.

The continued maturation of the Digital Twin concept has been driven by the convergence of several key enabling technologies. Foremost among these is the integration of artificial intelligence and machine learning, which elevate the Digital Twin from a passive mirror to an active predictive system capable of forecasting failures and optimizing performance through proactive maintenance. This predictive capability is supported by advances in real-time data fusion, combining inputs from onboard sensors, physics-based models, and historical datasets into a unified, continuously updated system representation. Such high-fidelity, data-

driven environments provide the ideal foundation for developing and validating the sophisticated logic required for autonomous operations, a capability of particular relevance to the IOS mission.

The operational significance of Digital Twin technology is best demonstrated through its application in pioneering space missions and within the commercial aerospace industry.

One notable European example is ESA's OPS-SAT mission, the first in-orbit laboratory dedicated to testing new mission control technologies. The OPS-SAT Digital Twin was not confined to pre-flight testing; it became an integral operational asset, enabling experimental software to be safely validated on the ground before uplink to the live spacecraft. This approach significantly reduced risk and enabled more than 280 successful in-orbit experiments, exemplifying the potential of Digital Twins to foster innovation in mission operations.

Another benchmark comes from NASA's Mars rover missions. At the Jet Propulsion Laboratory (JPL), a full-scale engineering model of the Perseverance rover, known as OPTIMISM, operates within a simulated Martian environment. Coupled with high-fidelity software models, this physical twin allows operators to test every command sequence, from mobility planning to robotic arm operations, before transmitting them to Mars. Given the inherent communication delays, this methodology has proven indispensable for ensuring mission safety and reliability in remote and inaccessible environments.

In the commercial aerospace sector, companies such as Boeing and Rolls-Royce have demonstrated the lifecycle value of Digital Twin technology. Boeing's T-7A Red Hawk program, for example, improved the first-time quality of manufactured components by over 40% through the use of integrated digital twins, while Rolls-Royce's IntelligentEngine initiative employs real-time digital twins for predictive maintenance, reducing downtime and optimizing engine performance. These examples underscore the growing role of Digital Twins in improving efficiency, safety, and overall system economics across the aerospace domain.

Building upon this established foundation, the work presented in this thesis synthesizes these state-of-the-art principles to address the novel challenges of autonomous in-orbit servicing. It aims to extend the remote operations paradigm, as exemplified by NASA's rover missions, to the validation of complex, autonomous robotic activities where direct human intervention is impossible. Furthermore, it

draws inspiration from ESA's OPS-SAT mission by employing a Digital Twin as a safe, in-orbit validation environment for deploying new capabilities throughout a mission's lifetime. Finally, by emphasizing standardization and reusability, this work aspires to deliver the same lifecycle benefits, spanning pre-flight verification, operational monitoring, and post-mission analysis, that have already proven transformative in commercial aerospace.

2.6 Synthesis and Technology Selection Rationale

The preceding review of standards, frameworks, and programming paradigms culminates in the technology selection that underpins this thesis. This selection does not represent a set of isolated technical decisions, but rather a coherent engineering strategy in which each component logically derives from the fundamental requirements of the project.

The strategy originates from the highest-level programmatic constraint: the need to ensure that the developed model constitutes a reusable, interoperable, and enduring asset within the European space ecosystem. This requirement directly mandates compliance with the ECSS-SMP standard. Adherence to this standard, in turn, necessitates the use of a compatible software ecosystem. Accordingly, the SIMULUS framework has been selected as the development and validation platform, given its role as ESA's reference implementation of the SMP standard, while the CNES ISIS framework has been identified as the operational deployment target.

With the standard and frameworks established, attention shifts to the implementation methodology. The challenge of representing a complex, modular satellite system naturally aligns with the principles of OOP, which provides a logical and scalable structure for modeling discrete, interacting subsystems. To realize this OOP architecture with the computational performance and deterministic behavior demanded by a mission-critical Digital Twin, C++ emerges as the definitive choice. Its compiled nature, low-level memory control, and maturity within aerospace applications deliver the efficiency and reliability essential for real-time and safety-critical operations.

In synthesis, the chosen technology stack embodies a structured response to the requirements of modern European space missions. ECSS-SMP ensures

portability and standardization; SIMULUS provides a professional, validation-ready development environment; OOP offers a logical and extensible modeling structure; and C++ delivers the computational rigor necessary for high-fidelity, real-time execution. Together, these technologies form a cohesive and future-proof foundation for developing a Digital Twin that is not only technically robust, but also sustainable, interoperable, and fully aligned with established best practices in the European space industry.

3 System Architecture and Simulation Environment

This chapter presents the architecture of the simulation environment developed for the design, integration, and validation of the IOS Digital Twin. The entire system is hosted on a Dell Pro Max Tower T2 workstation equipped with an Intel Core Ultra 9 285K processor, 128GB of RAM, and a 2TB solid-state drive, operating under Windows 11 as the primary system.

Built upon this physical infrastructure, the environment adopts a virtualized dual-machine architecture derived from the IOS Software Validation Environment (SVE). Implemented through a hypervisor, this setup ensures a realistic separation of concerns between the satellite simulation and the ground control subsystems.

A distinctive feature of the Intel Core Ultra 9 285K processor is its hybrid architecture, combining eight high-performance Performance-cores (P-cores) with sixteen energy-efficient Efficiency-cores (E-cores). Unlike many modern processors, this model does not support Hyper-Threading, a characteristic that introduces specific configuration challenges for simulation workloads. To prevent the host operating system from scheduling computationally demanding processes on the lower-performance E-cores, a processor affinity mask was applied. This configuration binds all hypervisor processes exclusively to the P-cores, ensuring that the virtual machines consistently access the highest-performing hardware resources. This measure is essential for maintaining the deterministic, real-time behavior required by the simulation.

The following sections describe in detail the configuration of this virtualized environment, the functional roles of each virtual machine, and the data flow mechanisms that govern their interaction.

3.1 Virtualized System Architecture

The simulation architecture is implemented using VMware Workstation Pro 17, enabling the creation of a self-contained, portable, and easily reproducible virtualized environment. This environment is structured around two distinct virtual machines (VMs) that replicate the operational separation between the spacecraft and its ground control segment. The first VM, referred to as the Simulation Host,

runs SUSE Linux Enterprise Server (SLES) 15.2, while the second VM, designated ECHO, serves as the Operator Console and operates under Windows 10.

To ensure stable and isolated communication, the two VMs are interconnected through a private VMware LAN Segment. This setup establishes a dedicated virtual network completely segregated from any external connectivity, thereby preventing interference and ensuring secure data exchange. Within this private LAN, both virtual machines are assigned static IP addresses, guaranteeing a predictable and reliable TCP/IP connection. In this client-server configuration, the ECHO VM functions as the client, initiating communication with the SIMSAT kernel running on the Simulation Host, which acts as the server and listens on a predefined network port.

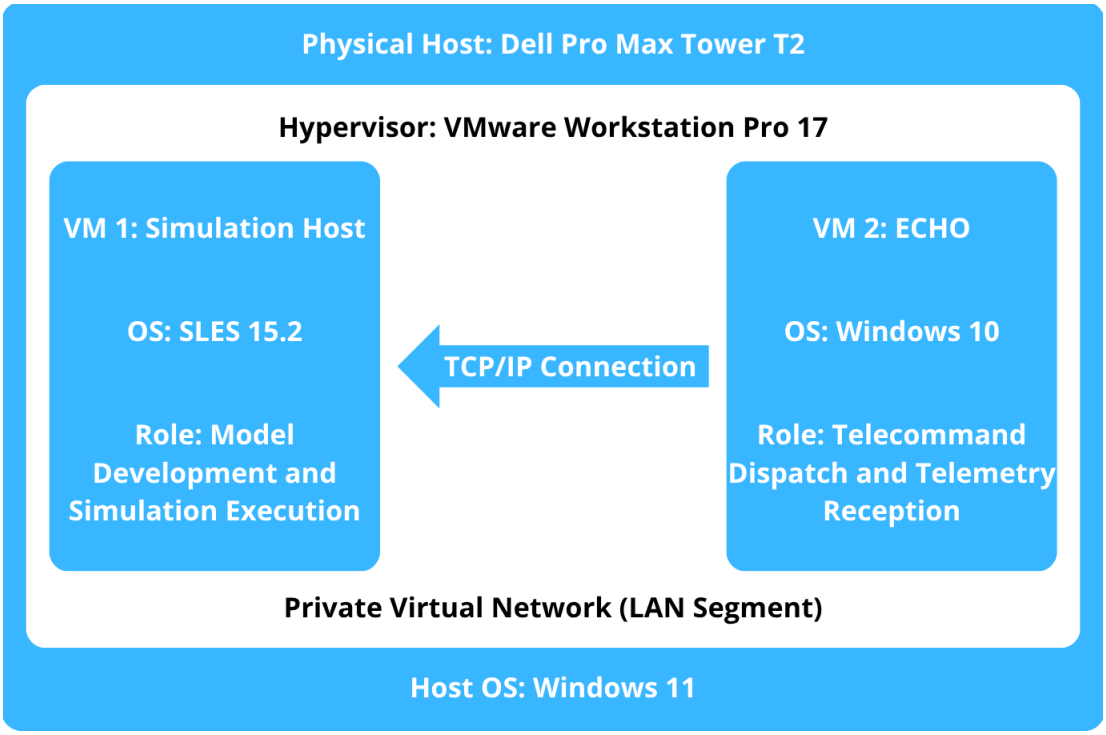

Figure 2: High-Level Architecture of the Virtualized Simulation Environment

A high-level representation of this architecture is provided in Figure 2, which illustrates the physical host containing the two virtual machines, their respective operating systems and software roles, and the private virtual network facilitating their interaction.

3.2 The Simulation Host VM

The Simulation Host serves as the computational core of the entire simulation environment. This virtual machine is specifically configured to deliver the performance, stability, and software ecosystem required for the development and execution of the IOS Digital Twin.

To accommodate the demanding computational load, the Simulation Host is provisioned with 16 virtual CPU cores, 64GB of RAM, and a 200GB virtual hard drive. The operating system, SLES 15.2, is employed as it is a mandatory prerequisite for the installation and reliable operation of the core simulation software.

The software environment deployed on this machine comprises the complete MDE toolchain necessary for Digital Twin development. This includes the SIMULUS 10.3.0 framework, Visual Studio Code as the primary Integrated Development Environment (IDE), and MagicDraw 2021x for model design. The latter tool is used to create the master blueprint of the model, previously described in Subsection 2.2.1, using the Unified Modeling Language (UML), a standardized graphical notation for specifying and visualizing software architectures.

A critical element of this setup is the Terma Emulator (TEMU), which provides high-fidelity, instruction-level emulation of space-grade processors based on the SPARC architecture, including the ERC32, LEON2, LEON3, and LEON4. For the IOS mission, TEMU is configured to emulate the LEON4 processor, which represents the spacecraft's OBC. Integrated within the SIMSAT environment, TEMU enables SITL validation by executing the actual flight software on the emulated processor. This capability allows the interaction between the onboard software and the simulated hardware models to be tested with a high degree of realism prior to deployment.

All project-related files are organized under a primary directory, /ios, on the virtual machine's file system. This directory is logically divided into two main subfolders:

- /ios/tasi.sve: Contains the development environment, including all build scripts, source code, and UML design files that form the master blueprint of the Digital Twin.
- /ios/Simulator: Serves as the runtime directory, containing the compiled libraries and configuration files necessary to launch and operate a simulation

session within the SIMSAT kernel.

In operational mode, this VM functions as the simulation server. An operator initiates this server from the command-line, which launches the SIMSAT kernel. The kernel then loads the compiled satellite model, pauses the simulation at T = 0, and begins listening for a TCP/IP connection from the Operator Console VM on a dedicated network port, ready to receive telecommands and transmit telemetry data in real time.

3.3 The ECHO VM

The second virtual machine, referred to as ECHO, serves as the Operator Console. Its primary function is to provide the HMI for interacting with the live simulation, effectively representing the ground segment within this virtualized architecture.

ECHO operates on a Windows 10 system and is configured with 8 virtual CPU cores, 32GB of RAM, and a 100GB virtual hard drive. It hosts a proprietary suite of Thales Alenia Space software applications developed for mission control and testing. These applications work together to manage the command and control loop. The main components of this suite are described below:

- Cortex Interface (iride_if.exe): This application acts as the primary communication gateway. It establishes and maintains separate TCP/IP client connections to the SIMSAT server for both the telecommand uplink and telemetry downlink. Its configuration file, iride_if.ini, defines the network parameters required to connect to the Simulation Host.

- On-Board Environment (OBE4.exe): This tool provides real-time telemetry visualization and monitoring. It receives, decodes, and displays telemetry packets from the SIMSAT kernel, allowing operators to assess the satellite's status. It serves as the main interface for confirming the reception and execution of telecommands through corresponding acknowledgment and status telemetry messages.

- Telecommand Console (Tcc.exe): This is the operator's main command interface. The Tcc enables "online" mode for command transmission and pro-

vides functionality for sending both predefined library telecommands and custom raw hexadecimal telecommands to the simulated spacecraft.

- Message Transport Protocol (Mtp): This background service manages the message transport layer required by the higher-level Tcc application. Proper operation of Mtp is essential for ensuring reliable command transmission.

Together, these applications form a cohesive command and control environment. They allow an operator on the ECHO VM to send telecommands, monitor the satellite's health and status in real-time, and verify the outcome of their actions, thus enabling a complete and realistic simulation of mission operations. The detailed workflow for using these tools to conduct a simulation run will be described in the following section.

3.4 Simulation Workflow and Data Flow

The dual-machine architecture described in the previous sections enables a realistic simulation workflow that closely replicates an actual mission operations scenario. This workflow can be divided into a series of phases, spanning from system initialization to the execution of the command and control loop, each governed by a well-defined flow of data between the two virtual machines.

The process begins with the initialization and handshake phase. On the Simulation Host VM, the operator first launches the SIMSAT kernel using a command-line script. At this stage, the kernel loads the satellite model and begins listening for incoming network connections, while the simulation time remains paused at T = 0. Next, on the ECHO VM, the operator starts the Cortex interface (iride_if.exe), which establishes the TCP/IP connections for both the telecommand and telemetry channels with the waiting SIMSAT kernel. A successful handshake is confirmed once the connection indicators turn green and corresponding connection messages appear in the SIMSAT log.

Once the network link is established, the operator sets up the ground control environment on the ECHO VM by launching the required applications, primarily the telemetry display (OBE4.exe) and the telecommand console (Tcc.exe), and bringing them to an online state.

The execution phase begins on the Simulation Host VM. Although the kernel is initiated via the command line, the simulation clock is started from the SIMSAT

Graphical User Interface (GUI). Starting the clock triggers the simulation, prompting the satellite model to begin generating a continuous stream of telemetry data. This marks the start of the primary telemetry flow: telemetry packets are produced by the SIMSAT kernel, transmitted across the virtual LAN, received by the Cortex interface on the ECHO VM, and finally decoded and displayed in real time by the OBE4 application.

With the simulation running and telemetry streaming, the operator can perform the core command and control loop. The data flow for a single command sequence unfolds as follows:

1. Telecommand Uplink: The operator composes and sends a telecommand using the Tcc application on the ECHO VM.

2. Telecommand Transmission: The telecommand packet is transmitted over the virtual network to the SIMSAT kernel on the Simulation Host VM.

3. Model Execution: The SIMSAT kernel routes the command to the relevant SMP interface of the satellite model, which processes it and updates its internal state accordingly.

4. Telemetry Feedback: In response, the model generates acknowledgment and completion telemetry packets (e.g., TM(1,1) for validation and TM(1,7) for execution).

5. Verification: These telemetry packets return to the ECHO VM via the established telemetry data flow and are displayed in the OBE4 event log. The operator confirms successful command execution by observing this feedback.

This workflow establishes a complete and interactive simulation loop, enabling thorough end-to-end testing of the Digital Twin. From the issuance of operator commands to the verification of model responses, the system provides a robust framework for validating mission operations within a controlled, virtual environment.

4 Model Development

4.1 The In-Orbit Servicing Electric Power System: Architectural Concept and Technologies

The EPS is a critical subsystem located within the Orbital Propulsion and Support Module (OSPM) of the IOS satellite. It is responsible for generating, storing, conditioning, and distributing the electrical power required by all subsystems across both the OSPM and the Avionics and Robotics Platform (ARP). As the IOS mission remains in the developmental phase, the architectural concept presented here represents the technical reference upon which the digital twin model is constructed. This reference has been synthesized from key project documentation, including system architecture concepts, avionics design definitions, and the technical requirement specifications of the primary EPS components.

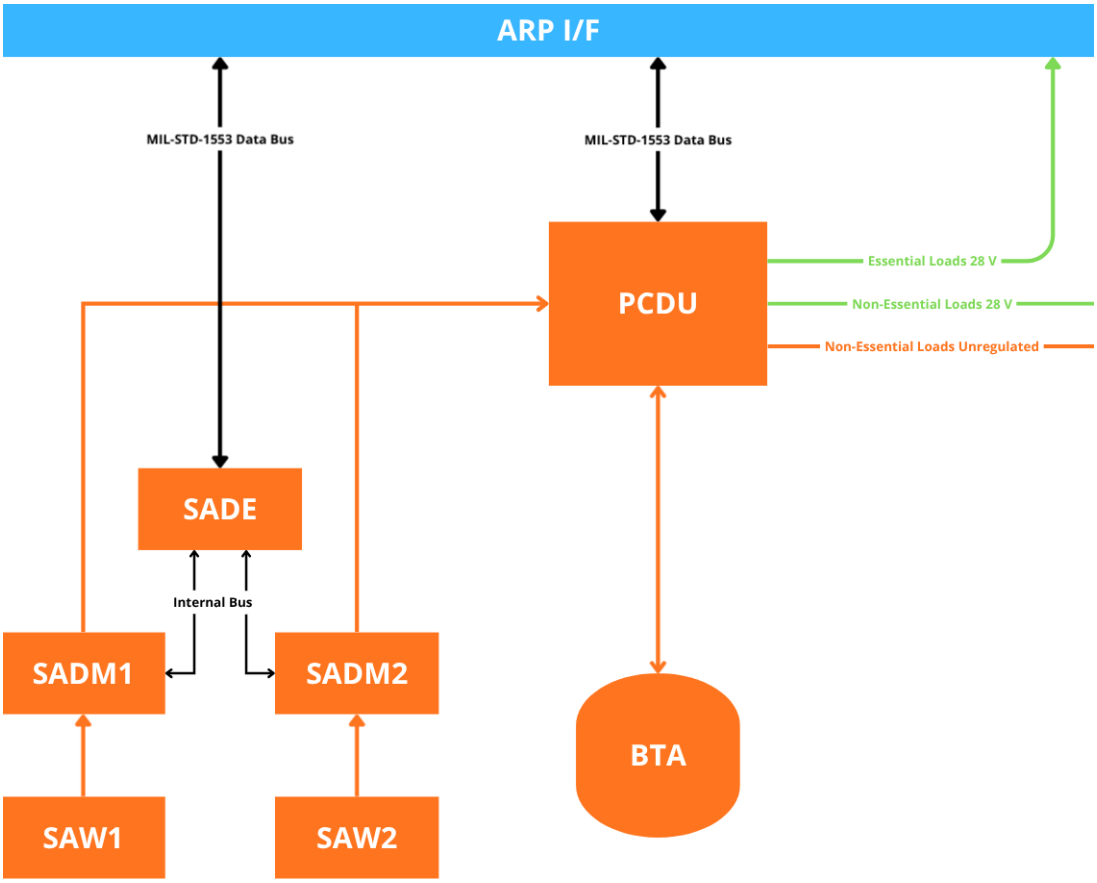

Figure 3: Architectural Concept of the Electric Power System

The EPS employs a hybrid power architecture managed by a central PCDU. At its core lies a main unregulated power bus, operating within a nominal voltage range of 45V to 63V, supplied directly by the solar arrays and the main battery. From this primary bus, the PCDU also generates a stable and fault-tolerant 28V regulated bus, which powers the spacecraft's essential avionics. The complete architectural concept of the EPS, including the interconnection of power generation, energy storage, and power distribution functions is illustrated in Figure 3.

The power generation function is provided by two identical SAWs, each consisting of four deployable panels equipped with triple-junction Gallium Arsenide (GaAs) solar cells. The photovoltaic assembly is highly sectioned to improve reliability and regulation efficiency. Each array is organized hierarchically, with 31 cells in series per string, 8 strings in parallel per section, and 2 sections per panel. In total, the system comprises 16 independent sections (8 per wing), each connected to a dedicated Sequential Switching Shunt Regulator (S3R) within the PCDU.

To maximize energy harvesting, each solar array wing is mounted on a SADA, which continuously orients the panels toward the Sun. The assembly consists of a central Solar Array Drive Electronics (SADE) unit that controls two Solar Array Drive Mechanisms (SADMs). The SADE functions as the intelligent controller of the system, receiving sun-pointing commands and transmitting telemetry data, both its own and that of the SADMs, via the MIL-STD-1553 data bus.

The energy storage function is provided by the main BTA, composed of two redundant packs built from Lithium-ion cells. Each pack is arranged in a 15-series, 24-parallel (15s24p) configuration, delivering a nominal capacity of 84Ah at Beginning-of-Life (BOL) and operating within the unregulated bus voltage range. The battery system is capable of a maximum charge rate of 42A and exhibits high round-trip efficiency, with charge and discharge efficiencies of 86% and 98%, respectively.

At the center of the EPS, the PCDU acts as both the intelligent controller and the functional core of the system, performing power conditioning and distribution. It regulates power flow from the 16 solar array sections via the S3R module, shunting excess power to maintain bus voltage stability. The Battery Charge Management (BCM) function controls the battery's State-of-Charge (SOC), implementing a Constant Current / Constant Voltage (CC/CV) charging algorithm. For

power distribution, the PCDU provides the 28V regulated bus and delivers power to all spacecraft loads, including essential and non-essential avionics, heaters, and propulsion units, through a network of fault-tolerant output lines protected by various types of Latching Current Limiters (LCLs). The PCDU is fully commandable, and its operational status is continuously monitored via the MIL-STD-1553 data bus interface.

4.2 Modeling Philosophy and Abstraction

The development of the EPS model followed a design philosophy emphasizing behavioral fidelity over detailed physical simulation. Since the parent satellite design is still under development, the primary objective was to produce a model that is functionally and behaviorally accurate at its external interfaces. The aim is to provide a sufficiently realistic representation of the EPS to serve as a foundation for future subsystem models, specifically enabling their connection to a standardized power line interface, rather than to perform in-depth physical analyses of the underlying hardware.

To achieve this, several abstractions were intentionally introduced to manage model complexity and ensure alignment with its intended purpose. Physical effects such as thermal dependencies on component performance, long-term degradation due to radiation or charge/discharge cycling, detailed internal electronic behavior, and high-frequency electrical transients were excluded from the current scope. The model instead focuses strictly on the logical, electrical, and communication behavior of the subsystem components.

The model's software architecture directly implements the SMP and OOP principles introduced in Chapter 2. Each major hardware component, SAW, SADA, BTA, and PCDU, is encapsulated in its own C++ class, with internal states and logic hidden from the outside. To ensure seamless integration with the host environment and facilitate code reuse, these classes inherit from standard SIMULUS base classes: GenericModel for core SMP functionality, GenericCyclicModel for components requiring periodic updates, and GenericRemoteTerminal for MIL-STD-1553 bus communication.

The model leverages polymorphism to handle state updates. Each component class implements a specialized version of the virtual PulseRaisingPort() method,

inherited from GenericCyclicModel. This method, periodically invoked by the SIM-SAT scheduler, contains the key logic responsible for updating the component's internal state. The system's interfaces follow the SMP data paradigm, in which configurable parameters are exposed as Properties and commandable functions as Operations. While the public interfaces were fully defined, the specific telemetry outputs were not yet implemented in this version, pending the definition of the higher-level software architecture.

The resulting engineering trade-offs prioritize the immediate utility of establishing a robust power distribution framework, enabling future models to be readily integrated and validated against the power line interface. For instance, whereas earlier simplified models treated certain subsystems as perpetually powered, the new EPS model provides a dynamic power distribution network, enabling future subsystem models to be connected to and interact with individual power lines. This functional approach provides a substantial improvement in validation capability, offering realistic testing conditions while maintaining flexibility to adapt as the hardware design continues to evolve.

4.3 The Solar Array Wing and Solar Array Drive Assembly Model

The power generation subsystem is modeled through two primary, interconnected components: the SAW model, which simulates the conversion of solar energy into electrical current, and the SADA model, which controls the orientation of the wings. This section presents the conceptual logic and C++/SMP implementation of both models.

4.3.1 The Solar Array Wing Model

Concept The SAW model provides a behaviorally accurate representation of the wing's electrical output. Its core logic is based on an empirically derived current-voltage (I-V) function, developed from End-of-Life (EOL) performance requirements under a comprehensive set of worst-case conditions. These conditions include minimum solar irradiance, extreme operating temperatures, a single-string failure, and radiation-induced degradation.

The power output of the model depends on three primary factors:

1. Bus voltage, which determines the current supplied according to the I-V function.
2. Eclipse state, modeled as a binary switch that alternates between sunlight and shadow phases of a simplified orbital cycle, enabling or disabling power generation accordingly.
3. Sun incidence angle, provided as a pointing error from the SADA model, which modulates the generated current using a cosine law to account for off-nominal pointing.

The model also incorporates key operational logic. The wing deployment sequence is explicitly represented, ensuring that no power is generated before the wing is released. Deployment is simulated by activating Non-Explosive Actuators (NEAs) upon receiving a valid command pulse from the PCDU, which triggers a release flag after a defined delay. Additionally, to support fault scenario testing, the model allows any individual solar string to be commanded into a failed state, removing its contribution to the total output current.

Implementation The SAW model is implemented using a hierarchical composite pattern that mirrors the physical hardware structure. The top-level SolarArrayWingModel acts as a container for instances of SolarPanelModel and GenericPyroModel (representing the NEAs). Each SolarPanelModel contains several SolarSectionModel instances, and each section, in turn, contains the lowest-level SolarStringModel objects. All subcomponents are instantiated and managed during the configuration phase.

- SolarStringModel: Inheriting from GenericModel, this class represents the fundamental power-generation element. It computes the output current by linearly interpolating values from a static I-V lookup table (defined in the SMP configuration file SawConfig.h) based on the input voltage. It also exposes an SMP operation, FailString(), to simulate string-level failures.
- SolarSectionModel and SolarPanelModel: Both classes, derived from GenericModel, serve as aggregators. Their primary function is to sum the currents

from their child components and pass the result upward through the hierarchy.

- GenericPyroModel: A pre-existing library class used to represent the NEAs. It is configured to respond to power pulses received through the IPowerPulseLineIF interface and manages the deployment timer.
- SolarArrayWingModel: The top-level class, inheriting from GenericCyclicModel, aggregates the total current from all panels, monitors deployment status via the contained GenericPyroModel instances, and applies the effects of eclipse state and sun incidence angle. Its main logic executes within the overridden PulseRaisingPort() method, which is periodically invoked by the SIMSAT scheduler. The model interacts with the PCDU through a standardized power line, instantiated as a subcomponent.

Integration with SMP is finalized in the assembly file (saw.xml), where the SAW_1 and SAW_2 instances are declared and their power line subcomponents are connected to the PCDU, demonstrating catalog-based integration within the SMP framework.

4.3.2 The Solar Array Drive Assembly Model

Concept Based on the preliminary documentation available, the SADA model is implemented as a simplified behavioral representation derived from key technical requirements: a minimum angular precision of ±1 ° and a constant minimum angular velocity of 0.01 °/s. The model receives a target angle and simulates rotation by updating its current angle at each simulation step until it falls within the specified tolerance band of the target position.

Implementation The SADA model follows a distributed architecture that cleanly separates functionality across distinct components. The top-level SadeModel acts as a container for a SadeRemoteTerminalModel, which handles bus communication, and maintains aggregation relationships with two SadmModel instances representing the physical drive mechanisms.

- SadeRemoteTerminalModel: Inheriting from GenericRemoteTerminal, this class serves as the interface between the model and the MIL-STD-1553

bus, managing low-level data exchange through the BcToRt() and RtToBc() methods.

- SadeModel: Inheriting from GenericModel, this class acts as the control unit of the SADA. It interprets commands received from the remote terminal, assembles telemetry packets (including those compliant with the Packet Utilization Standard (PUS)), and issues drive commands to the mechanisms.
- SadmModel: Inheriting from GenericCyclicModel, this class represents the physical mechanism. Its PulseRaisingPort() method performs the slew maneuver and includes logic for both nominal and redundant actuators. It computes the current pointing error and communicates this information to the SolarArrayWingModel through an SMP data flow field.

The assembly of the SADA model is defined in its sada.xml file. This file instantiates the both nominal and redundant electronics as SADE_N and SADE_R, creates a SADM model for each of the two SAWs and links each remote terminal to the MIL-STD-1553 bus model, ensuring a modular and realistic implementation of the complete drive assembly.

4.4 The Battery Model

The BTA model simulates the energy storage subsystem, which is responsible for supplying power during orbital eclipses and storing excess energy generated by the solar arrays.

Concept The core of the BTA model is designed to track its SOC over time using a coulomb-counting algorithm. At each simulation step, the Depth-of-Discharge (DOD) is updated by integrating the net current flowing into or out of the battery. This process explicitly incorporates the energy losses specified in the hardware documentation, applying a charge efficiency of 86% to incoming current and a discharge efficiency of 98% to outgoing current.

The Open-Circuit Voltage (VOC) is modeled as a linear function of the DOD, ranging from a maximum of 4.2V per cell pack at 0% DOD (fully charged) to a minimum of 3.1V per cell pack at 100% DOD (fully discharged). A key abstraction in this model is that the terminal voltage is assumed to be equal to the VOC.

Voltage drops due to internal or harness resistance are neglected to simplify the model while preserving an accurate representation of the overall energy balance. The model's state evolution is driven by the net current provided by the PCDU, which determines whether the battery is in a charging or discharging mode.

Implementation The BTA model is implemented using a hierarchical composite pattern, reflecting the modular structure of the physical battery. The top-level BtaModel class acts as a container that instantiates and manages a collection of BtaCellPack objects.

- BtaCellPack: Inheriting from the GenericModel base class, this component represents a single pack of parallel cells. Its main responsibility is to compute the pack's VOC based on its current DOD.
- BtaModel: The top-level class, derived from GenericCyclicModel, governs the behavior of the entire battery. Its main logic is executed in the PulseRaisingPort() method, which is called periodically by the SIMSAT scheduler. During each update cycle, the class iterates through all contained BtaCellPack instances, supplying them with the current value and commanding them to update their internal states. The resulting voltages are then aggregated to determine the total terminal voltage of the battery. The model's interaction with the PCDU is handled through a standardized power line, instantiated as a subcomponent.

Integration of the model into the broader simulation is defined in the assembly file (bta.xml). This file instantiates the top-level battery as BTA and specifies the connection between its power line subcomponent and the corresponding interface on the PCDU model. This configuration demonstrates the catalog-based architecture of SMP, which enables physical interconnection of components within the virtual simulation environment.

4.5 The Power Control and Distribution Unit Model

The PCDU forms the functional core of the EPS, acting as the central hub for energy management and distribution. Within the Digital Twin, the PCDU model is

the most complex component, as it governs the interactions between the power sources (SAW and BTA) and the spacecraft loads.

Concept The conceptual design of the PCDU model is built around a power balance algorithm that executes at each simulation step. The model first determines the total power available from the solar arrays and compares it with the total power demand from all active spacecraft loads. Based on this comparison, it dynamically establishes the operational mode of the EPS and manages the flow of energy to maintain the stability of the unregulated bus.

A central part of this logic is the BCM. When excess power is available from the solar arrays, the model initiates a battery charging sequence that follows a full CC/CV algorithm. During this phase, the battery is charged at its maximum allowable current until the terminal voltage reaches a predefined End-of-Charge Voltage (EOCV) threshold. At that point, the model transitions into a constant-voltage tapering phase, gradually reducing the charge current to prevent overcharging. Conversely, during eclipse periods or high-load conditions, the model identifies the resulting power deficit and draws the required current from the battery to maintain power supply continuity.

The behavior of the S3Rs within the solar array's interface is modeled abstractly: rather than simulating individual switching operations, any solar power not consumed by the loads or used for battery charging is treated as shunted and effectively discarded.

Finally, the PCDU model simulates power distribution by managing the state of individual switches. Each power line can be switched on or off via the MIL-STD-1553 bus, providing a commandable interface that allows future models to draw power from specific, controlled sources within the simulation.

Implementation The PCDU model employs a composite design pattern, in which the main Pcdu class acts as an orchestrator for several specialized subcomponents, each responsible for a distinct function. The top-level Pcdu class contains instances of Pcdu1553, PcduPowerSupplier, PcduBatPowerConsumer, and PcduSAPowerConsumer.

- Pcdu1553: Inheriting from GenericRemoteTerminal, this class serves as the exclusive interface between the PCDU and the MIL-STD-1553 bus. It

receives and decodes raw command words from the bus and translates them into executable actions, such as turning specific switches on or off.

- PcduPowerSupplier: Derived from GenericModel, this class manages the on or off state of all power distribution lines. It maintains an internal map of connected loads and, when commanded by the main Pcdu class, delivers either the regulated 28V or the unregulated bus voltage to the corresponding power line.
- PcduSAPowerConsumer and PcduBatPowerConsumer: Both inherit from GenericPowerConsumer and act as interface adapters. They implement the standardized IPowerLineIF interface, through which the main Pcdu class exchanges current and voltage data with the solar arrays and the battery.
- Pcdu: The top-level class, inheriting from GenericCyclicModel, coordinates all subsystem functions. Its primary logic resides in the PulseRaisingPort() method, called periodically by the SIMSAT scheduler. This method reads inputs from the consumer components, calculates the bus voltage and required battery current, and issues commands to the relevant sub-models. Key operational parameters, such as the battery's EOCV and maximum charge current, are defined in the SMP configuration file PcduConfig.h

The complete system integration is defined in the assembly file (pcdu.xml). This file not only instantiates the main Pcdu and its subcomponents, but also defines the critical interface links between the consumer classes and the SAW and BTA models, as well as the connection between the remote terminal and the MIL-STD-1553 bus model. This configuration demonstrates the modular, scalable, and fully SMP-compliant architecture of the implemented system.

5 Model Validation and Results

5.1 Validation Strategy and Test Procedures

This section presents the validation of the EPS model developed in this thesis. The primary objective of the validation campaign is not to demonstrate that the model is a perfect physical replica of the hardware, but rather to confirm that it is fit for purpose. In this context, "fit for purpose" means verifying that the model's behavior aligns closely with that of an authoritative high-fidelity reference and that its accuracy is sufficient for its intended application: to provide a stable and validated power source for the development and integration of future Digital Twin subsystem models.

Given that no physical flight hardware is currently available, the validation strategy relies on a comparative analysis against a trusted, high-fidelity reference: the official mission Power and Energy Budget document. This document contains the full set of performance specifications and expected data profiles for the EPS. Due to the confidential nature of this reference, the specific methods used to generate its data cannot be disclosed. Accordingly, the adopted strategy involves executing identical test scenarios on the C++/SMP model and comparing the resulting output against the reference data and the corresponding success criteria defined in the budget document.

The test cases used for this validation were selected from the comprehensive set of mission scenarios analyzed in the Power and Energy Budget. The selection was guided by the requirements established following the Preliminary Design Review (PDR), which specify two key operational profiles to be sustained by the EPS:

- A nominal cyclic phase, with an average load of 1800W for 36 minutes of discharge, and
- A special flight operation, with a load of 2100W sustained for 3000 seconds.

Two scenarios from the budget analysis were identified as representative of these profiles:

- Case 1-2a: A sequence simulating the early mission phases, from launch to a stable in-orbit condition, representing nominal cyclic operation.

- Case 6.2: A scenario simulating a rendezvous operation conducted entirely during eclipse, representing a high-power special phase.

To ensure a meaningful one-to-one comparison, a set of common initial conditions and modeling assumptions, derived from the reference document, was applied to the C++/SMP simulations. These included the simulation of a single solar array string failure per section and the use of a constant average sun incidence angle of 28.7°, with no active SADA rotation commanded. Furthermore, the power demand for each phase was modeled as a constant average value, without explicit switching of individual loads. During each simulation run, key output variables were recorded to data files for post-processing and analysis.

The model's performance was evaluated using both qualitative and quantitative criteria. Qualitatively, the time evolution and overall trends of the outputs generated by the C++/SMP model, such as battery State of Charge (SOC) and bus voltage, were compared visually against the corresponding reference plots from the Power and Energy Budget. Quantitatively, the model results were assessed against the formal success criteria defined in the reference analysis. The most critical of these criteria requires that the minimum bus voltage at the PCDU interface remains above 50V under all test conditions.

5.2 Test Cases

All validation tests described in this chapter were executed within the specific hardware and software environment detailed in Chapter 3. The test campaigns were performed on the Dell Pro Max Tower T2 workstation, which hosts the dual virtual machine architecture via VMware Workstation Pro 17. This configuration consists of a SUSE Linux-based Simulation Host VM, running the SIMULUS 10.3.0 framework and the C++/SMP model, and a Windows-based Operator Console VM used for command and control. The following subsections describe the specific test cases conducted within this environment to validate the behavior of the EPS model against the reference data provided in the Power and Energy Budget document.

5.2.1 Nominal Operation: Test Case 1-2a

The first test case validates the performance of the EPS model during a nominal multi-phase operational sequence, with emphasis on the overall power balance and the charge/discharge dynamics of the battery across multiple orbits. This scenario is based on Case 1-2a as defined in the mission's Power and Energy Budget document.

The simulation begins with the spacecraft in its post-launch configuration and progresses through five distinct mission phases:

- Phase 1: Ground Chronologies & Launch
- Phase 2: Launcher Separation to SA Deployment (Sun Acquisition)
- Phase 2b: SA Deployment to Target Release
- Phase 3: In-Orbit Phase (S-band Tx only)
- Phase 4: In-Orbit Phase (S-band + X-band Tx)

The initial conditions are defined with the SAWs stowed (no power generation) and the battery initialized at a voltage of 61.5V, corresponding to its EOCV. The spacecraft power demand follows a representative mission profile: moderate during launch, increasing after separation, and peaking during the target release phase. Thereafter, the power consumption stabilizes at a sustained high operational level for the in-orbit phases, as summarized in Table 1.

	Phase 1	Phase 2	Phase 2b	Phase 3	Phase 4
Power Demand [W]	805	1058	1806	1609	1650
Duration [s]	4760	3181	6144	55200	55200

Table 1: Power Demand and Duration for each Phase of Case 1-2a

According to the Power and Energy Budget reference data, the expected outcome of this test is a dynamic evolution of the EPS state. The solar arrays remain inactive during Phases 1 and 2, with the spacecraft drawing exclusively from the battery, resulting in a substantial initial discharge. Upon solar array deployment at the start of Phase 2b, power generation begins, initiating a sequence of charge/discharge cycles synchronized with the sunlit and eclipse portions of

each orbit. In the subsequent steady orbital phases (3 and 4), the SOC of the battery is expected to oscillate between approximately 90% at the end of sunlit periods and 65% at the end of eclipse. Correspondingly, the unregulated bus voltage should remain well within operational limits, stabilizing near the EOCV during sunlight and dropping to roughly 56V during eclipse, safely above the 50V minimum requirement.

5.2.2 Special Operation: Test Case 6.2

The second test case validates the EPS model under a high-power, battery-dominant operational scenario, simulating a critical rendezvous sequence executed entirely within an orbital eclipse. In this condition, the spacecraft must rely solely on the battery for power supply. The scenario is derived from an aggregation of the high-power "special phases" defined in the Power and Energy Budget document.

The simulation progresses through the following five mission phases:

- Phase 18: S4 Hold (a nominal in-orbit phase)
- Phase 19: Approach/Distancing Near range (S4) to Proximity (S5)
- Phase 19b: Approach/Distancing Proximity (S5) to Ready 4 Docking
- Phase 20: Capturing & Berthing
- Phase 21: Mated

A defining aspect of this scenario is that Phases 19 through 20, the most power-intensive maneuvers, occur consecutively within a single eclipse period. The initial conditions are set with the solar arrays fully deployed, but with the battery initialized at a reduced SOC of 45% to simulate a stressed pre-operation state. The power demand profile begins with a nominal load during the hold phase and increases sharply through the approach and capture phases, reaching its maximum during the "Capturing & Berthing" phase, as summarized in Table 2.

	Phase 18	Phase 19	Phase 19b	Phase 20	Phase 21
Power Demand [W]	1828	2024	2172	2451	1689
Duration [s]	53040	660	300	1200	55200

Table 2: Power Demand and Duration for each Phase of Case 6.2

According to the Power and Energy Budget reference data, the expected outcome of this test is a rapid, deep discharge of the battery during the eclipse-bound operations. The battery current is expected to peak at approximately –50A during the highest power-draw phase. Consequently, the battery SOC is predicted to decrease steeply, reaching a minimum of about 50% by the end of the berthing phase. The unregulated bus voltage should follow a similar trend, falling from its nominal level to a minimum of approximately 52V. Despite this significant discharge, the voltage is expected to remain above the 50V mission success threshold, thereby confirming the system's ability to sustain high-demand operations under eclipse conditions.

5.3 Discussion of Results and Model Limitations

This section presents the results obtained from the C++/SMP model for the two test cases defined previously. The outputs from the model, illustrated in the figures below, are compared against the reference data and performance criteria defined in the Power and Energy Budget document. This comparison forms the basis for assessing the model's performance and validating its suitability for its intended purpose.

5.3.1 Nominal Operation: Test Case 1-2a

The results for the nominal operations scenario, shown in Figure 4, demonstrate a strong correlation with the reference data, particularly during the initial discharge phases. The load profile and sun/eclipse flag are accurately synchronized with the test conditions. During the initial battery-only period (Phases 1 and 2), the model correctly reproduces the expected discharge trend, with the Battery SOC, Battery Current, and Bus Voltage curves closely matching the reference behavior.

A noticeable deviation appears at the onset of the first charging phase (Phase 3). The C++/SMP model predicts a charge current approximately 10% higher than the reference, resulting in a faster battery recharge. The battery reaches its EOCV within roughly four orbital cycles, compared to six cycles in the reference data. This accelerated charging is reflected in the SOC and Bus Voltage trends. Nevertheless, during the subsequent stable eclipse phases, the model's predictions for minimum SOC (50%) and minimum Bus Voltage (53V) remain quan-

titatively consistent with the reference, confirming the model's accuracy during discharge conditions.

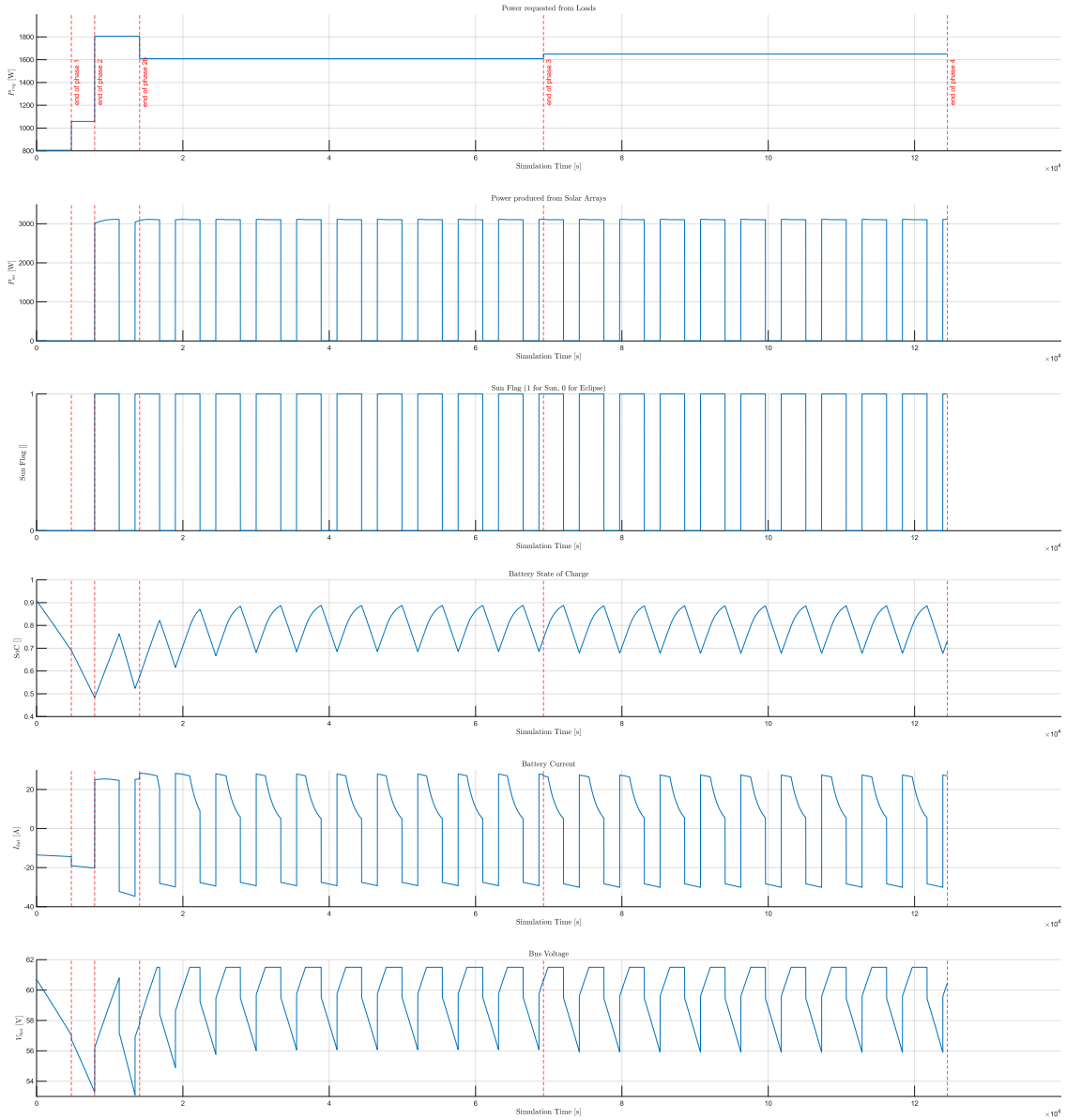

Figure 4: C++/SMP Model Simulation Results for Test Case 1-2a

5.3.2 Special Operation: Test Case 6.2

A similar behavior is observed in the high-power rendezvous scenario, shown in Figure 5. During the initial "S4 Hold" phase, the model's higher charge rate

again results in a faster approach to the EOCV, reaching it within seven orbital cycles, whereas the reference data indicates no full charge within the ten-cycle test window.

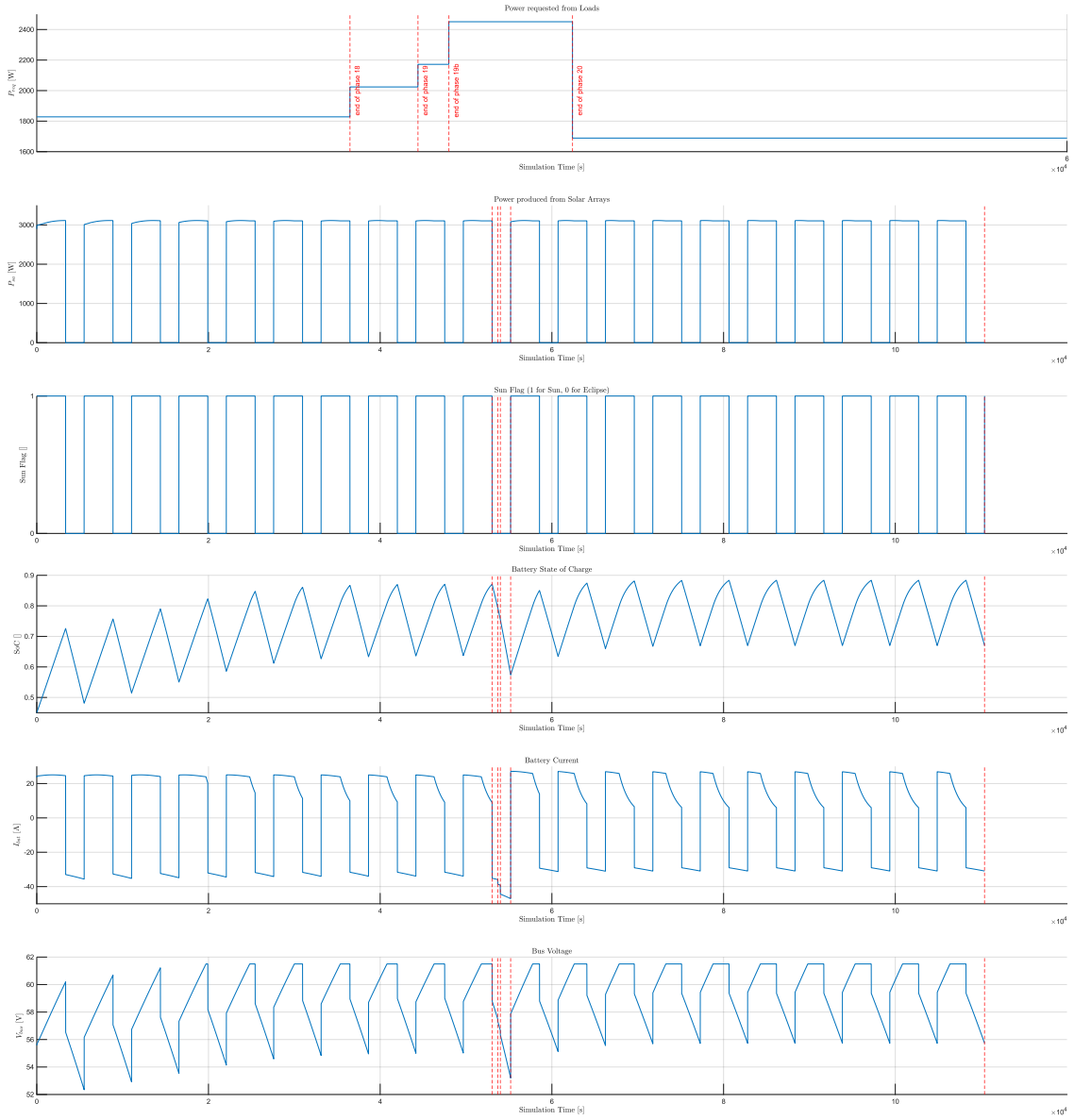

Figure 5: C++/SMP Model Simulation Results for Test Case 6.2

The main focus of this test, however, lies in the high-power discharge that occurs during the rendezvous and berthing maneuvers. In this phase, the model demonstrates excellent correlation with the expected behavior. The simulation accurately reproduces the steep drop in SOC and Bus Voltage associated with the

high load demand. Quantitatively, the results are well aligned with the reference: the peak discharge current reaches nearly –50A, the minimum SOC at the end of the maneuver stabilizes around 50%, and the minimum Bus Voltage remains safely above the mission threshold at approximately 52V.

5.3.3 Analysis of Discrepancies

The validation results reveal a consistent pattern: the model performs with high accuracy during discharge phases, while a systematic discrepancy appears during charging periods. This difference originates from the simplified representation of the S3R logic within the PCDU.

In the current implementation, the C++/SMP model applies an idealized power balance algorithm. At each simulation step, it calculates the total available solar power, subtracts the spacecraft load demand, and assumes that any surplus power can be instantaneously and perfectly directed to the battery, constrained only by the battery's voltage-dependent acceptance rate.

In contrast, a real S3R operates through sequential switching of solar array sections and exhibits finite control-loop response times. These dynamics introduce small inefficiencies and transient effects that slightly reduce the effective average charge current delivered to the battery. The reference data, which captures these real-world behaviors, therefore shows a slower charging response. The simplified approach used in the C++/SMP model leads to a theoretically optimal charge transfer at each time step, resulting in the observed approximately 10% faster charge rate. Importantly, this discrepancy is confined to charging behavior; the strong agreement during discharge phases confirms the validity of the core battery model.

5.3.4 Model Limitations and Validation Conclusion

The validation campaign confirms the robustness and operational realism of the EPS Digital Twin while also highlighting the following limitations:

- The predictive accuracy during battery charging phases is reduced due to the idealized S3R regulation logic.

- Thermal and degradation effects on component performance are not modeled.
- The simulation operates in discrete time steps and does not capture high-frequency electrical transients, making it unsuitable for analyses of phenomena such as inrush currents during load activation.
- The eclipse and sun-pointing models are based on simplified, hardcoded values and do not account for real-world orbital perturbations.

Despite these limitations, the EPS model is considered validated and fit for purpose. The results demonstrate that it accurately reproduces the behavior of the system during the mission's most critical operational regimes, nominal eclipse operation and high-power discharge scenarios. Since these conditions represent the dimensioning cases for the satellite's power system, the proven fidelity of the model in these areas confirms its suitability as a reliable power source for testing the behavior of future interconnected models under nominal and special load conditions. The identified limitations, particularly in the charging model, do not undermine this purpose but rather define clear directions for future refinement and fidelity enhancement.

6 Conclusion

This final chapter synthesizes the work undertaken in this thesis. It begins by summarizing the main objectives, the adopted methodology, and the principal outcomes of the project. It then provides a critical reflection on the limitations of the developed model and the lessons learned from the engineering trade-offs made during its development. Finally, it outlines a strategic roadmap for future work, positioning this thesis as a foundational step toward the creation of a comprehensive, operational Digital Twin for the IOS mission.

6.1 Summary of Work

This thesis addressed the critical need for a standards-compliant simulation environment to support the verification and validation of flight software for the IOS mission. The work was conducted within a rigorous, industry-aligned framework to ensure that the resulting Digital Twin would be not only technically robust but also reusable and interoperable across future European space programs.

The selected technological foundations directly reflected these objectives. The ECSS-SMP standard was adopted to ensure model portability and a clear separation between the model and its simulation environment. A dual-machine virtual architecture was configured using the ESA-reference SIMULUS framework, establishing a professional ecosystem for model development and validation, with the CNES ISIS environment identified as the final operational deployment platform. The subsystem models were implemented in C++ following an OOP paradigm, providing the performance, determinism, and modularity required to accurately represent a complex spacecraft power system.

The core contribution of this work is the design and implementation of a functional, behaviorally accurate model of the satellite's EPS, encompassing the SAWs, SADA, BTA, and PCDU. The model was successfully integrated into the virtualized simulation environment, enabling full end-to-end testing from a ground-operator console. Validation against reference data from the official Power and Energy Budget document confirmed that the model is fit for purpose and sufficiently accurate for its intended use as a foundational component for the integration and testing of future Digital Twin models.

6.2 Limitations and Lessons Learned

A central design philosophy of this work was to prioritize behavioral fidelity over deep physical simulation, a pragmatic decision reflecting the early stage of the IOS mission's hardware design. This approach enabled the rapid delivery of a useful, functional model, but also introduced a number of limitations that provide valuable lessons for future development.

The most significant limitation, identified during the validation campaign, concerns the reduced accuracy of the model during battery charging phases. This discrepancy arises from the idealized abstraction of the S3R logic within the PCDU. The key lesson learned is a clear example of an engineering trade-off: simplifying the control logic accelerated model delivery and reduced complexity but introduced a deviation in non-critical operational regimes. The strong accuracy observed during discharge phases confirms that this was an acceptable trade-off, as those scenarios represent the dimensioning cases for power system performance, thereby providing a robust baseline for evaluating the power consumption of future integrated models.

Other limitations stem from the defined modeling scope. The exclusion of thermal dependencies, long-term degradation effects, and high-frequency electrical transients means the model represents the logical and electrical behavior of the EPS, not its full physical dynamics. This distinction reinforces an important lesson: the value of a model is determined by its purpose. By focusing strictly on the behaviors relevant to power generation and distribution, the model achieves its intended utility as a stable and reliable foundation for integrating other subsystem models, without unnecessary complexity. These limitations do not detract from its engineering value but instead define its valid operating envelope and highlight clear directions for further refinement.

6.3 Future Work and Outlook

This thesis establishes a solid foundation for the continued development of the IOS Digital Twin. The next steps can be organized as a roadmap for progressive fidelity enhancement and functional expansion.

The immediate priority is to refine the regulation logic of the PCDU to resolve the charging-current discrepancy identified during validation. As the physical

satellite design evolves, the model should undergo iterative fidelity upgrades to maintain synchronization with its hardware counterpart. A key enhancement will be the integration of thermal models, leveraging the capabilities of the SIMULUS framework to enable a coupled electro-thermal simulation. In the longer term, the EPS model should be integrated with other subsystem models, such as GNC and AOCS, to enable holistic, end-to-end spacecraft simulations.

Ultimately, the long-term vision for this work is the connection of the validated model to the real-time telemetry stream of the in-orbit satellite following launch. Achieving this would complete the transition from a ground-based validation tool to a true, operational Digital Twin, enabling real-time health monitoring, predictive diagnostics, and on-the-fly procedure validation during the mission. This thesis thus represents the first and essential step toward that vision, delivering a sustainable and high-value engineering asset for the future of in-orbit servicing.

References

[1] B. D. Allen, “Digital twins and living models at NASA”, NASA Technical Reports Server (NTRS), Nov. 03, 2021. https://ntrs.nasa.gov/citations/20210023699

[2] M. Proietti, “Firmato il contratto ASI-Thales Alenia Space per la prima missione italiana di In-Orbit Servicing”, ASI, May 16, 2023. https://www.asi.it/2023/05/firmato-il-contratto-asi-thales-alenia-space-per-la-prima-missione-italiana-di-in-orbit-servicing

[3] Q. Kong, T. Siauw and A. Bayen, Python programming and Numerical Methods: A Guide for Engineers and Scientists. Academic Press, 2020.

[4] P. Zipfel, “A C++ architecture for unmanned aerial vehicle simulations”, AIAA Infotech@Aerospace 2007 Conference and Exhibit, May 2007, doi: 10.2514/6.2007-2945.

[5] “Joint Strike Fighter Air Vehicle C++”, QA Systems. https://www.qa-systems.com/solutions/joint-strike-fighter-air-vehicle-jsf-av-c

[6] “ECSS-E-ST-40-07C Rev.1 - Simulation Modelling Platform - Level 1”, European Cooperation for Space Standardization, Aug. 05, 2025. https://ecss.nl/standard/ecss-e-st-40-07c-rev-1-simulation-modelling-platform-level-1-5-august-2025

[7] “SIM website”, SIMULUS project. https://sim.space-codev.org

[8] J.-M. Georger, “CNES-ISIS, Design of future ground system operations over a fully automatized stack of CCSDS-MO Services”, 2018 SpaceOps Conference, May 2018, doi: 10.2514/6.2018-2573.

[9] “Mission OPS-SAT”, ESA. https://esoc.esa.int/content/ops-sat

[10] N. Hartono, “NASA Readies Perseverance Mars Rover’s Earthly Twin”, NASA, Sep. 20, 2023. https://www.nasa.gov/centers-and-facilities/jpl/nasa-readies-perseverance-mars-rovers-earthly-twin

[11] “Let’s Connect: Digital Thread Advances manufacturing”, Boeing. https://www.boeing.com/innovation/innovation-quarterly/2022/10/let-s-connect-digital-thread-advances-manufacturing

[12] B. Goldenberg, “Driving Innovation: Rolls Royce’s Success with Digital Twins,” ISM, Mar. 05, 2025. https://ismguide.com/rolls-royce-use-of-digital-twin-technology-case-study

[13] “Digital Twins: Accelerating aerospace innovation from design to operations,” Airbus, Apr. 23, 2025. https://www.airbus.com/en/newsroom/stories/2025-04-digital-twins-accelerating-aerospace-innovation-from-design-to-operations