Contributions to satellite subsystem models and communications link emulation

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Antonio Romero Aguirre

In partial fulfilment

of the requirements for the degree in

TELECOMMUNICATIONS TECHNOLOGIES AND SERVICES ENGINEERING

Advisor: Adriano José Camps Carmona

Joan A. Ruiz-de-Azua

Barcelona, June 2022

Abstract

During last years, space missions have evolved from monolithic spacecrafts architectures, to distributed ones, which are known as Distribute Satellite Systems (DSS). This spacecraft architecture has changed the space paradigm, as it has appeared the need of establishing communications between satellites to achieve the mission objectives. Despite this the need of simulating and emulating the communications between satellites, none of the available tools have complete inter-satellite communications capabilities.

This final degree thesis presents contributions to the already existing simulator developed by NanoSat Lab. The contributions consist in the implementation of the Electric Power Supply (EPS) subsystem. Moreover, this thesis also contributes to the development of a Satellite Contact Emulator, developed by i2Cat, focusing on the development of the central PC that orchestrates the emulation.

Resum

Recentment, les missions espacials han evolucionat d'una arquitectura basada en un sol satèl·lit, a una arquitectura distribuïda. Aquest tipus d'arquitectura és coneguda com a *Distribute Satellite Systems"* (DSS). L'aparició d'aquesta nova estructura ha provocat un canvi en el paradigma de l'espai, ja que ha sorgit la necessitat d'establir comunicació entre els satèl·lits per assolir els objectiu marcats de les missions. Tot i haver la necessitat d'emular i simular les comunicacions enter satèl·lits, cap de les eines actuals té suficients funcionalitats implementades per assolir aquesta emulació i simulació.

Aquest treball de fi de grau presenta les contribucions fetes a un simulador ja existent, el qual ha estat desenvolupat pel NanoSat Lab de la UPC. La contribució feta consisteix en la implementació d'un subsistema que modeli una EPS. A més a més, aquest treball de fi de grau també contribueix en el desenvolupament d'un *Satellite Contact Emulator*, el qual ha estat desenvolupat per i2Cat. En aquesta contribució es farà menció especial en l'ordinador central que executa l'emulació.

Resumen

Recientemente, las misiones espaciales han evolucionado de una arquitectura basada en un solo satélite, a una arquitectura distribuida. Este tipo de arquitectura es conocida como *Distribute Satellite Systems"* (DSS). La aparición de esta nueva estructura ha provocado un cambio en el paradigma espacial, ya que ha surgido la necesidad de establecer comunicación entre los satélites para conseguir los objetivos de las misiones. Aunque exista la necesidad de emular y simular las comunicaciones entre satélites, ninguna de las actuales herramientas tiene las funcionalidades suficientes para conseguir dicha emulación y simulación.

Este trabajo de fin de grado presenta las contribuciones hechas a un simulador ya existente, el cual ha sido desarrollado por el NanoSat Lab de la UPC. La contribución hecha consiste en la implementación de un subsistema que modela una EPS. Además, este trabajo de fin de grado también contribuye al desarrollo de un *Satellite Contact Emulator*, el cual ha sido desarrollado por i2Cat. Esta contribución pone un énfasis especial en el ordenador central que ejecuta la emulación.

Acknowledgements

I would like to thank the thesis advisors, Adriano Camps and Joan A. Ruiz-de-Azua the support showed towards me. Moreover, I would like to also thank them the given opportunity to discover the space sector, which was mostly unknown by myself before starting this thesis.

Also, I would like to recognize the Nanosat Lab team for the received treatment, as well as the i2Cat Space Communications group. Special mention also for Nanosat Lab and i2cat institutions for letting me use their installation to develop this final degree thesis.

Revision history and approval record

Revision	Date	Purpose
0	15/03/2022	Document creation
1	25/05/2022	Document revision
2	11/06/2022	Document revision
3	18/06/2022	Document revision
4	21/06/2022	Document final revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Antonio Romero Aguirre	antonio.romero.aguirre@estudiantat.upc.edu
Adriano José Camps Carmona	camps@tsc.edu
Joan A. Ruiz-de-Azua	joan.ruizdeazua@i2cat.net

Written by:		Reviewed and approved by:			
Date	21/06/2022	Date	21/06/2022	Date	21/06/2022
Name	Antonio Romero	Name	Adriano Camps	Name	Joan A. Ruiz-de-Azua
Position	Project Author	Position	Project Supervisor	Position	Project supervisor

Table of contents

List of Figures

List of Tables:

Acronyms

ADCS	Attitude Determination and Control System
COTS	Commercial off-the-shelf
DSS	Distributed Satellite System
DSS-SIM	Distributed Satellite System Simulator
eMSM	Extended Modified Shepherd Model
EPS	Energy Power System
IoSat	Internet of Satellites
IoT	Internet of Things
ISL	Inter-Satellite Link
LEO	Low Earth Orbit
MSM	Modified Shepherd Model
NB	Narrow Band
NB-IoT	Narrow Band-Internet of Things
NS-3	Network Simulator 3
SCE	Satellite Contact Emulator
SDP	Simplified Deep Space Perturbation
SDR	Software Defined Radio
SGP	Simplified General Perturbation
SoC	State of Charge
SoH	State of Health

1. Introduction

Monolithic satellites have been ruling the space by providing a custom design that accomplishes a specific mission. However, this kind of satellites where only one spacecraft is deployed have some drawbacks associated with them, mainly related with re-visit time and coverage range (depending on the orbiting plane). To mitigate the limitations associated with monolithic spacecrafts, the concept evolved to Distributed Satellite Systems (DSS), which are networks of heterogeneous or homogenous satellites with a common mission. Six main architectures of DSS can be distinguished according to [1], constellations, satellite trains, clusters, satellite swarms, and innovative mission concepts such as fractionated spacecraft and Federated Satellite Systems (FSS). Some of these architectures perform communication with other spacecraft in order to avoid collisions, such in the case of clusters or swarms, or to exchange data, as in the case of fractionated spacecraft or FSS.

This performed communication is known as inter-satellite links (ISL), which are point to point communications between two or more satellites. As exposed in [2],[3]; when ISLs have a routing protocol associated, it is known as Internet of Satellites (IoSat). This change in the paradigm has created the need of tools that help engineers to design and test the new communication protocols derived from IoSat needs. However, currently there are not many tools available that performs these operations, as it will be discussed in Sections 2.1 and 2.4, so i2CAT[1] Space Communications research group is currently developing two projects to fill the void on that field.

First of the projects was born at NanoSat Lab[2], which is an initiative of the CommsSensLab[3] research center of the Department of Signal Theory and Communications, with the support of the Barcelona School of Telecommunications Engineering. It is research laboratory focused on the exploration of innovative small spacecraft system concepts and developing and integrating subsystems and payloads notably for Earth Observation. The project is a simulation engine [4] known as Distributed Satellite System Simulator (DSS-SIM) that aims the design and testing of DSS interconnected by ISL. A more accurate description of how it works is performed at Section 3.1.

The other project held at i2Cat is a Satellite Contact Emulator, which aims to provide an interface to emulate contacts between satellites for NB-IoT and ISL scenarios. This project started from scratch with the development of this thesis, so its development is still in an early phase. However, a more description od the projects is done in Section 3.2.

[1] i2CAT - The Internet Research Center
[2] NanoSat Lab
[3] CommSensLab

1.1. Objectives

This final degree thesis aims the to address the lack of tools to simulate and emulate ISLs by contributing to the already existing DSS-SIM and creating a Satellite Contact Emulator (SCE).

The contributions to the first topic (extensions of DSS-SIM) will be:

1. The implementation of a module that simulates energy storage devices, in particular batteries or supercapacitors.
2. The implementation of a module that models solar cells, as the energy generation of spacecrafts.
3. The implementation of a module that implements an electric power system (EPS) for a satellite

The contribution to the second topic is:

1. Creation of a prototype that allows communications between two end-devices, connected into a network composed by Software Defined Radios (SDR) and a central PC that performs the ISL emulation.

It must be mentioned that the SCE is developed together with the student Arnau Dolz. This final degree thesis presents the central PC of the network, which propagates orbits and computes the parameters that affect the communications channels, such as attenuation, Doppler effect, and delay. Dolz final degree thesis [5] is focused on how the SDR processes the data, and how the communications channel is executed in the SDR from the parameters computed by the central PC.

1.2. Thesis outline

As explained, this final degree thesis presents two different projects, but both have the same motivation. Because of that, some of the chapters of the thesis will be divided into two subsections, where each of the subprojects are explained. The general structure is:

1. **Introduction:** Provides an overview of the project, where the rationale of the final degree thesis is clearly described
2. **State of the art:** Section provides a review of the current technologies to address the presented objectives is done. A review is conducted to assert the motivation of the projects.
3. **Methodology and development phases:** Section is divided into the two major constituents of this final degree thesis. The designed and implemented solutions for each, as well as the project phases are explained in this section.
4. **Tests and results:** As Section is divided into the two projects., and the performed tests and the resulting results are presented.

5. **Conclusions and future development:** Section presents a review of the initial objectives is done, explaining if they have been met or not for each of the projects. An overview of the future development of the projects is also performed.

The management features of the project are presented into the appendices, in order to keep the main body of the text just focused on the technical development. Appendices follow the next structure:

A. **Management:** The development phases of the projects are defined in detail, defining work packages, and including the initial and the final work plans, and their difference. Also, the list of requirements for each project is presented in this appendix.
B. **Budget:** The economic impact of the projects is considered here, including direct and indirect costs.
C. **Environmental impact:** A study of the projects' impact on the environment is performed at this appendix.

2. State of the art of the technology used or applied in this thesis:

2.1. Review of existing ISL simulators

Typically, the simulation of spacecrafts has been done using some licensed software, such as the well-known Systems Tool Kits[4] (STK). However, the space sector has suffered a remarkable change in recent years, which has increased the interest on ISL communications. Unfortunately, STK, as well as similar products, do not consider the capability of simulating communications protocols at ISL, so the tool falls short when addressing the simulation of DSS. To achieve the desired behaviour of simulators, there is the need to use network simulators. Some well-known open-source ones are Network Simulator 3 (NS-3)[5], OPNET[6], NetSim[7] or QualNet[8].

As it is clear, there is the need of a tool that combines both software in order of simulating a complete DSS into just one software. According to [4], three approaches can be done. One approach consists of running first the space simulator to obtain the desired parameters and use them as inputs for the network simulator. Another approach is the integration of both simulation engines into a single simulator that runs them in parallel. This approach was chosen by NASA [6], where they used QualNet and STK. Despite the promising approach, the resulting core resulted inefficient, because the constant communication between both simulation engines adds delay into the execution.

Finally, a third approach, and the most interesting one because of its versatility and resource management, is a fully integrated simulator. This approach consists of the implementation of a custom simulator that, with just one simulation engine (and non-using different software at same time) all the simulation parameters can be computed. In [7], a simulator based on NS-3 is presented. This project demonstrates the feasibility of the approach, despite it failed at demonstrating the re-usability of the software to simulate instrument and resources defined by the user.

To overcome the problems found in other approaches, a simulation tool is presented in [4]. The DSS-SIM is a simulation engine that aims to simulate the impact of ISL over spacecraft resources, enable the implementation of high-level spacecraft interactions and provide an adaptable tool where users can define physical models that represent spacecraft components and instruments. The simulator is constructed over NS-3, which manages the simulation of a defined scenario by the DSS-SIM. NS-3 is a free and open-source discrete-event network simulator that also provide some communication protocols which can be used in in the ISL context.

[4] Systems Tool Kit (STK)
[5] NS-3
[6] OPNET
[7] NetSim
[8] QualNet

2.2. Review of existing solar cells models

In space, the most accessible energy from environment is solar power, and the modules in charge of converting the sun irradiance into electricity at the solar cells. Its behaviour is difficult to describe because it is based on how electrons of the semiconductor material interact with hitting photons from solar cells [8]. Therefore, there are some models that describe its behaviour in simpler ways.

According to [9], the most used models are (1) the single diode mode, (2) the double diode model, (3) the modified double diode model, and (4) the three-diode mode. Nevertheless, the most widespread model is the double diode. The problem of this model is its complexity, as it takes a long simulation time to compute its parameters because of the differential relations between them. Because of that, researchers agree [9] that the use of a single diode model is sufficient to describe the behaviour of a solar cell, despite its lower accuracy. The single and double diode models can be found described at [10], [11], from where the next figure and equations are extracted:

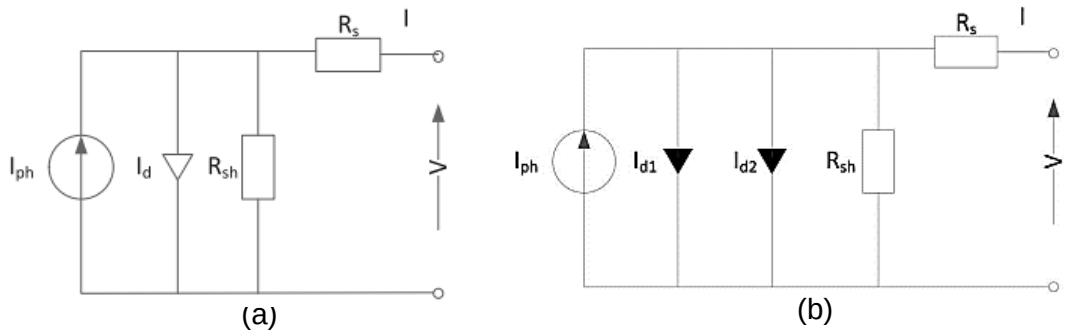


Figure 1: Single and double diode model

The equations that describe the single are:

$$I = I_{ph} - I_d \cdot (e^{\frac{V+I\cdot R_s}{n\cdot V_t}} - 1) - \frac{V + I \cdot R_s}{R_{sh}} \quad \textbf{(1)}$$

$$I_{ph} = I_{ph_0} \cdot \frac{I_r}{I_{ro}} \quad \textbf{(2)}$$

where:

- I_{ph} = Solar-induced current
- I_r = Light irradiance (W/m^2)
- I_{ph_0}= Measured generate current for irradiance I_{r0}
- I_d = Saturation current of diode

- $V_t = \frac{k \cdot T}{q}$ = Thermal voltage, where
  - k = Boltzmann constant
  - T = Temperature of operation
  - q = Charge on an electron
- N = Quality factor of the diode
- R_s = Resistor in series
- R_{sh} = Shunt resistor

2.3. Review of existing battery models

In the previous section it has been explained how spacecrafts obtain energy from the environment. As this energy is external to the spacecraft, there is no possibility to control the generated power. Because of that, there is the need to store the unused energy in batteries for a later use. As in the solar cells case, the exact equations of how a battery charges and discharges are difficult to obtain because they are based on chemicals processes that take place inside it. So, there is a need to simplify those equations by approximations known as models.

Battery modelling is useful as it can predict batteries basic parameters such as the current state of charge (SoC) or state of health (SoH) of the battery. Depending on the desired application, the accuracy of these parameters can be critical. In the DSS-SIM case, determining the SoC and SoH are important, but not critical, as the main objective of the simulation is the communications. However, it is important to represent how them impact the power modules. Typically, three types of models are distinguished: electrochemical, analytical/empirical, and circuit based [12].

The electrochemical models are based on the chemical processes that take place in the battery. Although these models describe the battery processes very accurately, they also have a higher degree of complexity than other methods because they might include high-order differential equations and the need of chemical parameters to describes battery's dynamics, which are usually difficult to obtain. Another limitation associated to these models is that, because of their complexity, it can take long simulation time to compute the parameters of interest. This limitation is really strong, as it only makes the models suitable for applications where high accuracy is needed, and makes them undesirable for the rest of applications, since many times speed is preferred over accuracy.

In Electric Circuit Models (ECM), the electrochemical reactions are replaced by circuits components such as resistors or capacitors that emulates the relation between the input parameters, SoC and current, and the voltage in terminals of a battery. Usually, the resistors represent the internal self-discharge of a battery, while RC networks represent the diffusion process in electrolyte and porous electrodes and the charge transfer in the electrode [12]. This type of models gives the developer a high degree of freedom designing a custom model for a desired application. Despite that, obtaining the parameters to emulate the relation between an ECM and an electrochemical model is complex. Some typical models are:

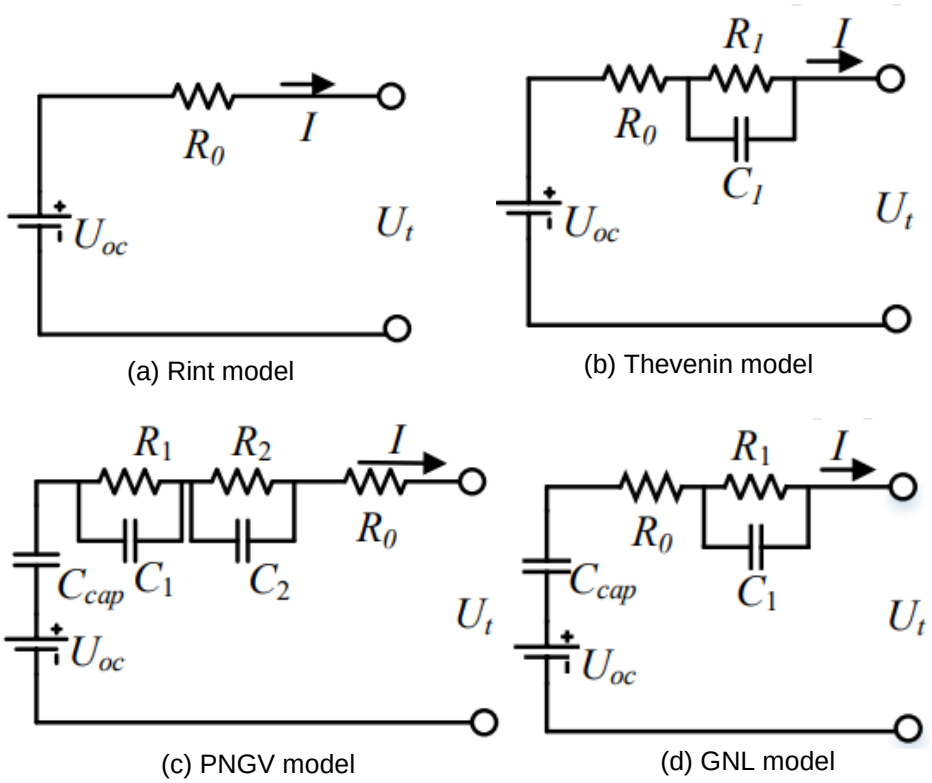


Figure 2: Different ECM [12]

The analytical or empirical models are considered as a simplified electrochemical model. In this model the high-order differential equations from electrochemical models disappears, and it are replaced by reduced order polynomials or mathematical expressions. This type of model leads to simpler expressions, which also is more efficient in terms of simulation time. The limitation is that accuracy is reduced because the expressions are approximations of the original one.

Considering the strengths and limitations of the different kinds of battery models, this case of study will only consider some of the analytical models.

2.3.1. Shepherd model

In 1965, Shepherd [13] presented a battery model designed from the charge/discharge curves:

$$V = V_0 - K \cdot \frac{Q}{Q - \int i \cdot dt} \cdot i + A \cdot e^{-B \cdot \int i \cdot dt} - R \cdot i \quad \textbf{(3)}$$

where:

- V = Battery voltage (V)
- V_0 = Battery constant voltage (V)
- K = Polarisation voltage (V)
- Q = Battery capacity (A·h)
- $\int i \cdot dt$ = Actual battery charge (A·h)
- A = Exponential zone amplitude (V)
- B = Exponential zone time constant inverse $(A \cdot h)^{-1}$
- R = Internal resistance (Ω)
- i = Battery current (A) (Positive if discharging, or negative if charging)

The first term of the equation represents the constant battery voltage. The second one is the current density plus the polarisation of the cell. The third one represents the effect at the beginning of the discharge curve, where the voltage drop is exponential. The last one is the voltage drop due to the internal resistance of the cell.

On one hand, the benefits of this model are that it is also an easy model in terms of computing and implementation, as the equations that describe the model are not differential or complex expressions. Also, the model is suitable for all battery technologies, as it has been demonstrated later [14], despite when it was proposed some of them did not exist. Moreover, the model is described using just one equation, only the current's direction changes, which simplifies its use.

On the other hand, the second term of the equation is non-linear, which adds some programming issues, such an algebraic loop, which is a closed loop where outputs are directly dependent on their inputs. If an algebraic loop exists, the simulation gets stuck because none of the components in the loop can generate output to break it. Also, the model does not consider some factors that affect the capacity of the battery, such as SoH or the external temperature. The strongest limitation is that the model is not well suited to describe the dynamic behaviour of batteries.

2.3.2. Copetti model

After reviewing some proposed models, Copetti [15] presented a new model, with the coefficients adapted to the current state of the technology at the time the model was presented. Copetti proposed the following equations:

Charge equation

$$V_C = [2 + 0{,}16 \cdot SOC] + \frac{1}{C_{10}} \cdot \left(\frac{6}{1 + I^{0{,}86}} + \frac{0{,}48}{(1 - SOC)^{1{,}2}} + 0{,}036\right) \cdot (1 - 0{,}025 \cdot \Delta T) \quad \textbf{(4)}$$

Discharge equation

$$V_C = [2{,}085 - 0{,}12 \cdot (1 - SOC)] - \frac{1}{C_{10}} \cdot \left(\frac{4}{1 + I^{1,3}} + \frac{0{,}27}{SOC^{1,5}} + 0{,}02\right) \cdot (1 - 0{,}007 \cdot \Delta T) \quad \textbf{(5)}$$

where:

- C_{10} = 10 hour rated capacity
- $\Delta T = T - T_{Ref} = T - T_{25℃}$

On one hand, the benefits of this model are that it is an easy model in terms of computing and implementation, as the equations that describe the model are not differential or complex expressions. Also, the model considers external factors such as temperature to describe the dynamics of the battery.

On the other hand, this model is only suitable for lead-acid batteries. Also, the model uses two equations to describe the behaviour of the battery, which is not desired for our application.

2.3.3. Modified Shepherd model

Although the Shepherd model is very interesting, the problem of the algebraic loop makes it not suitable for most of types of batteries. To correct that, Tremblay, Dessaint and Dekkiche [14] proposed a modified Shepherd model (MSM), which is the following one:

$$V = V_0 - K \cdot \frac{Q}{Q - \int i \cdot dt} + A \cdot e^{-B \cdot \int i \cdot dt} - R \cdot i \quad \textbf{(6)}$$

where:

- V = Battery voltage (V)
- V_0 = Battery constant voltage (V)
- K = Polarisation voltage (V)
- Q = Battery capacity (A·h)
- $\int i \cdot dt$ = Actual battery charge (A·h)
- A = Exponential zone amplitude (V)
- B = Exponential zone time constant inverse $(A \cdot h)^{-1}$
- R = Internal resistance (Ω)
- i = Battery current (A) (Positive if discharging or negative if charging)

As it can be seen, the term $K \cdot \frac{Q}{Q-\int i \cdot dt} \cdot i$ of equation **(3)** has been substituted by $K \cdot \frac{Q}{Q-\int i \cdot dt}$ in order to avoid the algebraic loop. Although this new term does not represent the behaviour of the battery as accurately as the original one, it can be use without a significant accuracy loss.

The benefits and limitations of the model are the same as the previous ones, but with this model the algebraic loop issue has been avoided.

2.3.4. Extended modified Shepherd model

The same authors as 2.3.3 in modified their own algorithm some years later and presented the extended modified Shepherd model (eMSM) [16].

Charge equation for Lead-Acid, NiMH and NiCd technology

$$V = V_0 - R \cdot i - K \cdot \frac{Q}{i \cdot t - 0{,}1 \cdot Q} \cdot i^* - K \cdot \frac{Q}{Q - i \cdot t} \cdot i \cdot t + e^t \qquad \textbf{(7)}$$

Discharge equation for Lead-Acid, NiMH and NiCd technology

$$V = V_0 - R \cdot i - K \cdot \frac{Q}{Q - i \cdot t} \cdot (i \cdot t + i^*) + e^t \qquad \textbf{(8)}$$

Charge equation for Li-Ion technology

$$V = V_0 - R \cdot i - K \cdot \frac{Q}{i \cdot t - 0{,}1 \cdot Q} \cdot i^* - K \cdot \frac{Q}{Q - i \cdot t} \cdot i \cdot t + A \cdot e^{-B \cdot i \cdot t} \qquad \textbf{(9)}$$

Discharge equation for Li-Ion technology

$$V = V_0 - R \cdot i - K \cdot \frac{Q}{Q - i \cdot t} \cdot (i \cdot t + i^*) + A \cdot e^{-B \cdot i \cdot t} \qquad \textbf{(10)}$$

where:

- V = Battery voltage (V)
- V_0 = Battery constant voltage (V)
- K = Polarisation voltage (V)
- Q = Battery capacity (A·h)
- $i \cdot t = \int i \cdot dt$ = Actual battery charge (A·h)
- A = Exponential zone amplitude (V)

- B = Exponential zone time constant inverse $(A \cdot h)^{-1}$
- R = Internal resistance (Ω)
- i = Battery current (A) (Positive is discharging or negative if charging)
- i^* = Filtered current (A)

On one hand, the benefits of this model are that it is a more accurate model than the previous versions, and it is suitable for all battery technologies, despite using different equations to describe the different modules.

On the other hand, the model introduces the concept of filtered current, which is not trivial to understand and obtain. Also, the model does not consider some factors that affect the capacity of the battery, such as the SoH, or the external temperature. However, its strongest limitation is that it uses different equations for charge and discharge, and it also uses different equations depending on the technology of the battery.

2.4. Existing SCE

The problem of being at an early stage of a technology development is that not many tools are available, as it's the case of satellite channel emulators. Three main branches have tried to assess the lack of products in this field: researchers, commercial brands, and governmental organizations.

An interesting approach in the research field is done in [17]. This approach consists of a set of SDRs interconnected through Ethernet, and also connected to Pc where the channel emulation is performed. However, this paper is based on the use of LabView, which is a paid software.

Currently, there exists some commercial alternatives such as S8825A[9] Channel Emulation Toolset developed by Keysight, or SLE900[10] Satellite Link Emulator developed by dBm Corp. However, this type of products has two main drawbacks associated:

- Cost: Commercial SCE are not affordable, as the cost is very expensive [18].
- Scalability: Commercial SCE have limited a limited input and output ports, which makes them not suitable for testing big satellites networks using just one of them.

Besides the research and commercial alternatives, some governmental organizations had also focused on the development of SCE. European Spatial Agency (ESA) inside ARTES framework, has developed two projects relate SCE. The projects are Real-time Satellite Network Emulator[11] and Emulator of Satellite-Terrestrial 5G Radio Channels[12], that are currently being developed. The architecture of the first project consists of three mains components:

[9] S8825A Channel Emulation Toolset
[10] SLE900
[11] Real-time Satellite Network Emulator
[12] Emulator of Satellite-Terrestrial 5G Radio Channels

- **STK**: Performs the design of the satellite system, as well as the orbital simulation (generation of satellite movement, visibilities, and link budget).
- **NS-3**: Simulates application traffic and related communication protocols on top of the satellite constellation.
- **Orchestrator**: Generates the STK scenario configuration files for the network simulator.

Also, NASA has some projects to achieve the channel emulation. An interesting one is the presented at [19]. This project aims to integrate a software toolset, which can be considered as a simulation, with hardware toolset, which is an emulator.

Once reviewed the available tools, it can be said that none of them fulfils the requirements for the SCE project. The institutional and commercial approach have the cost limitation, while the solution based on SDR uses a determined software to perform the operation.

2.5. Orbital models

Propagators are models that their objective is determining the position of a spacecraft at any instance of time given its initial state, which includes acceleration and velocity. According to Newton's laws, the motion of a body is determined by its initial state and the forces that act on it. Regarding that, if spherical Earth is assumed, the only force that acts on the system is the gravity, so the problem would be easy to solve. But real world is none like that, and the motion of a body is affected by some other factors such as Earth oblateness, gravitational fields from other celestial bodies or atmospheric drag. So, to determine orbits, there exists many models, that can e classified into numerical, semi-analytical and analytical [20].

On one hand, analytical models compute the analytical equations that describe the motion to obtain the final position. As these equations have some terms are complex to obtain, so many integrable expression are approximated. This approach leads into a reduced accuracy, but in the other hands helps with simulation times. On the other hand, numerical models compute the equations that describe the model without any approximation, which increase the accuracy of the model while sacrificing speed during computation. Also exists a third approach which is a combination of both. Semi-analytical models approximate some expression and compute the other ones. Its accuracy is less than numerical but have a higher performance.

Some of them are [21]:

- **Two body elements:** This model assume there are only two bodies in space, the Earth and orbiting satellite, and the only force that acts on the system is the gravitational force acting between them.
- **J2:** This model considers the perturbations introduced in the orbit due to Earth oblateness.
- **J4:** Considers the same effect (Earth oblateness) as J2 propagator but compute them with more coefficients (first and second order effects of J2 and the first-order effects of J4), leading into a more accurate model.

- **SGP:** Simplified General Perturbations (SGP) were developed by Hilton and Kuhlman in 1966 and is used for near-Earth satellites. This model assumes that the eccentricity is low, and that the perigee's altitude is constant.
- **SGP4:** Is an evolution of SGP, used for near-Earth satellites, when period is less than 225 minutes. It considers secular and periodic variations due to Earth oblateness, solar and lunar gravitational effects, gravitational resonance effects and orbital decay using a simple drag model.
- **SDP4:** Is an extension of SGP4 to be used for deep-space satellites when the period is superior of 225 minutes.
- **SGP8:** The SGP8 model propagator considers same effects as SGP4, but the calculation methods are different, which results in more accurate results.
- **SDP8:** The SDP8 model is an extension of SGP8 to be used for deep-space satellites. The deep-space effects are modelled in SDP8 with the same equations used in SDP4, but the calculations methods are more accurate.

3. Methodology and project development:

This section presents the design of both developments. Two sections are defined, corresponding to each of the developed projects. In the first subsection, a brief explanation of how DSS-SIM works will be done, as well as a detailed explanation of solar cells modules and battery modules, and how they will interact in order to model the behaviour of an EPS. In the second subsystem, an overview of designed SCE structure will be done to truly understand the project, and a more detailed explanation of the modules implemented by myself will be done. As it has been previously said, this second project is carried out between Arnau Dolz and me.

3.1. Contributions to DSS-SIM

An overview of DSS-SIM operations can be done by taking a look into its workflow diagram Figure 3. There three main phases can be distinguished during the simulation:

- **Simulator scenario:** It is built from user configuration files, where it is defined all parameters that describe the behaviour of each node. The configuration is done by three differentiated files type: component configuration, satellite definition and system definition. First of the files contains the description of custom satellite classes and their internal behaviour (e.g., description of battery parameters). Second type of files describe the spacecrafts model. In these files are described the networking components, such as protocols stack. Finally, last type of configurators describes the global system configuration, by determining orbit propagator characteristics for each node or the structure of the nodes (if they are inside a constellation or a monolithic satellite) With this approach of configuration,

users can model heterogeneous systems with both heavy multi-instrument satellites and single instrument small spacecraft.

- **Execution of the simulation scenario:** The simulator scenario is run by NS-3. Two main metrics are computed during the emulation: the position of the spacecrafts, which it is updated depending on the defined resolution, and event-based metrics (such transmission between nodes), which depend on the computed position of the spacecrafts.
- **Results of the simulation:** Two operations for data processing are performed during this stage. One of them fetches the data to visualize it using external tools, while the other operation analyzes system metrics across time.

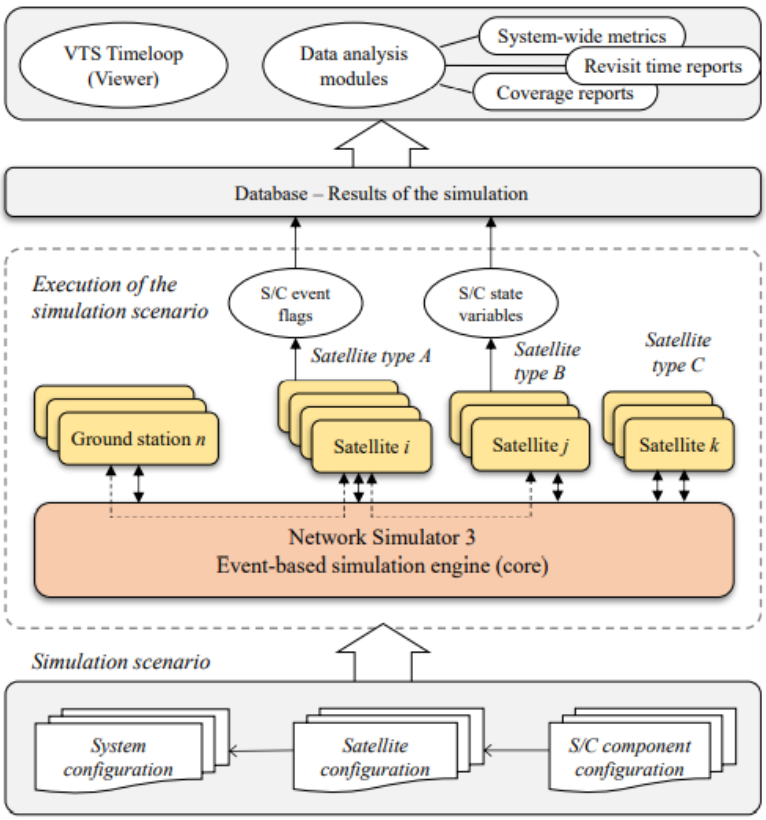


Figure 3: DSS-SIM workflow

The integration of new modules that describe the behaviour of a spacecraft shall be done using the class *Model*, which is inside the physical module (Figure 5), as this class provides the necessary tools for a correct implementation. Next is going to be done a brief explanation of the already implemented *Model* class, as well as all the different classes that interact with it. How these classes interact with each other is shown at Figure 6.

- *Model*: Generic abstract class to implement spacecraft components. Models are used to represent spacecraft states, physical components or devices, subsystems,

or payloads. This class is provided to simulator users that wish to model their specific components.

- *ModelConfig*: Provides options and the list of parameters, it also manages them. This class is used to pass the model parameters.

- *ModelFactory*: Generic abstract class to implement spacecraft components. It returns a new instance of a *Model*. It has the method *create()* to create a new model.

- *MInput*: Templated class that represents the input to a Model. It stores a smart pointer that points to an output of a *Model* object from which it reads its value.

- *MOutput*: Templated class that represents the output of a *Model*. The output of the model contains a value of type T and a Boolean flag that indicates if the value is set or not.

- *MLinkedIVariable*: An externally accessible input variable of a *Model*. *MLinkedIVariables* are connected to other variables to allow model objects to read the outputs and provide inputs to other models.

- *MLinkedOVariable*: An externally accessible output variable of a *Model*. *MLinkedOVariables* are connected to other variables to allow model objects to read the outputs and provide inputs to other models.

- *MState*: Templated class that stores a state variable of a *Model*.

A more specific explanation of how the class *Model* is shown with the workflow diagram at Figure 4.

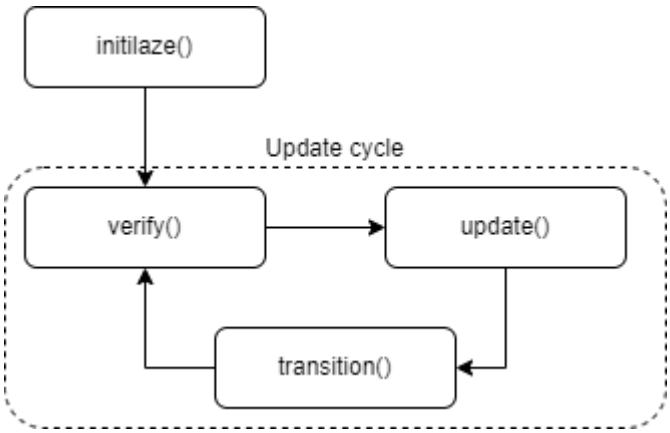


Figure 4: DSS-SIM *Model* workflow

First *initialize()* is called. As it names suggests, this function initializes the *Model* object. It is called once, before starting the *update cycle* loop. Model-derived classes should initialize the state of their variables and internal states in this function, while externally accessible variables will be linked right after the execution of this function.

Once the Model is initialized, *update cycle* loop begins. Custom *Models* awaits until one of its variables or one of its linked variables detect a change. When this occurs, it starts the update cycle, which consists of four function: *transition()*, *verify()* and *update()*. The first function transitions the *Model*'s states into its next state. Once this is done, the method *verify()* is called to ensure that changed input variables are safe to read, in order to not read a variable that has not been changed yet. Then the system calls *update()*, which is the main function of the *Model* object. This method reads all its input variables, perform some operations to them and update the values of its output variables. Moreover, the function can be also invoked by a self-update event. A more detailed explanation of this module is done at [22].

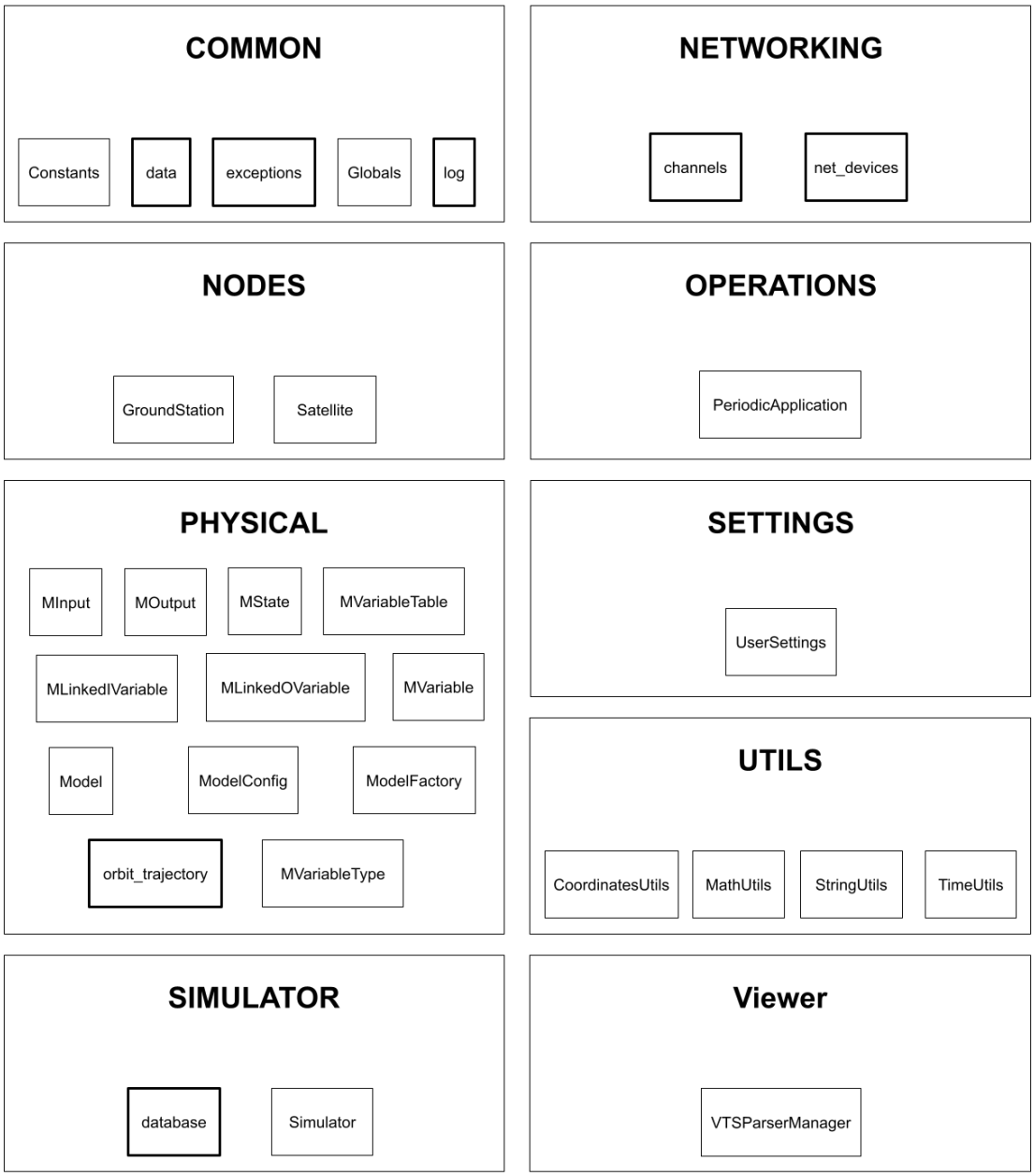


Figure 5: High-level architecture of DSS-SIM

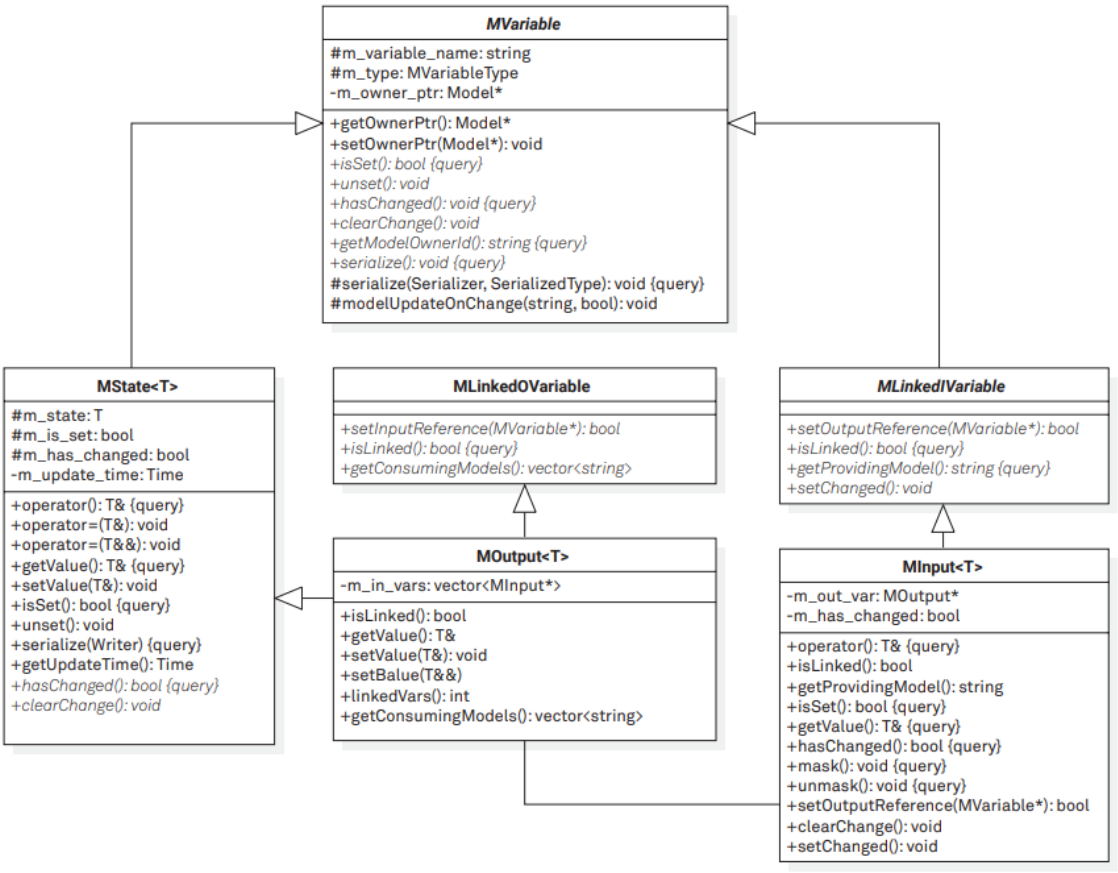


Figure 6: UML diagram of physical module

3.1.1. NS-3 energy framework

Now that the main module of the project has been presented into more detail, it is important to explain more accurately the NS-3. Its main functionalities and uses had already been explained in the previous section, so now a revision of the already implemented modules that concerns our case of study is going to be done. In [23] it is proposed an energy framework for NS-3 that consisted of two modules: an energy source that represents the power supply of a network node, and a device energy model, which implements the energy consumption model. Later, in 2014, Tapparello, Ayatollahi and Heinzelam [24] extended the energy framework, and its main contribution was the integration of an energy harvester, which represents the collection of energy from external resources. Some other contributions had been done to the framework during the following years, but none of them changed the structure in a significant way. The current energy framework structure is presented at Figure 7.

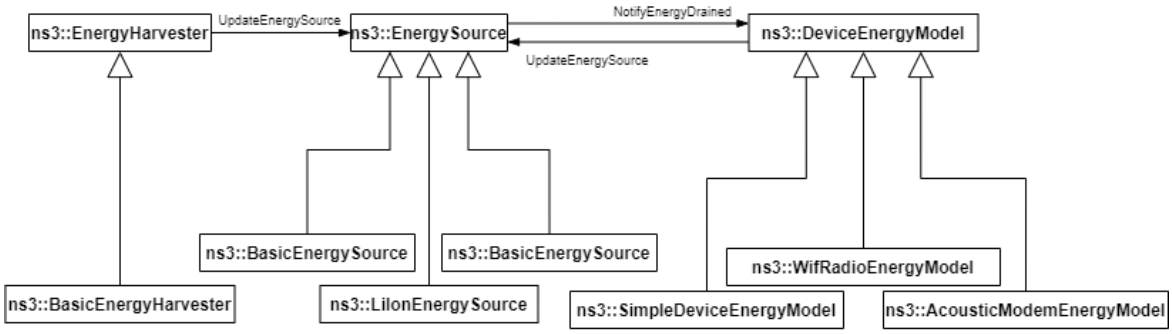


Figure 7: NS-3 Energy framework

As it can be seen in Figure 7, and as it has been briefly explained, NS-3 Energy Framework consists of three main modules. NS-3 API[13] defines each of the modules as:

- The Energy Source represents the power supply of each node. Connecting an energy source to a node where a device energy model had also been connected, implies that the corresponding device draws power from the source. On the other hand, connecting an energy source to a node where an energy harvester implies that the corresponding harvester generates power to the source. The basic functionality of the Energy Source is to provide energy to devices of the node. When energy is completely drained from the Energy Source, it notifies the devices of the node so that each device can react to this event. This module has implemented a battery based on [25], [26]. This model has not been taken into account in Section 2.3 because, despite it uses just 2 parameters, they are not easily obtainable.

- The Device Energy Model is the energy consumption model of a device installed on a node. It is designed to be a state-based model where each device is assumed to have several states, and each state is associated with a power consumption value. Whenever the state of the device changes, the corresponding Device Energy Model will notify the Energy Source of the new current draw of the device. The Energy Source will then calculate the new total current draw and update the remaining energy.

- The Energy Harvester represents the elements that collect energy from the environment and recharge the Energy Source to which it is connected. TS.

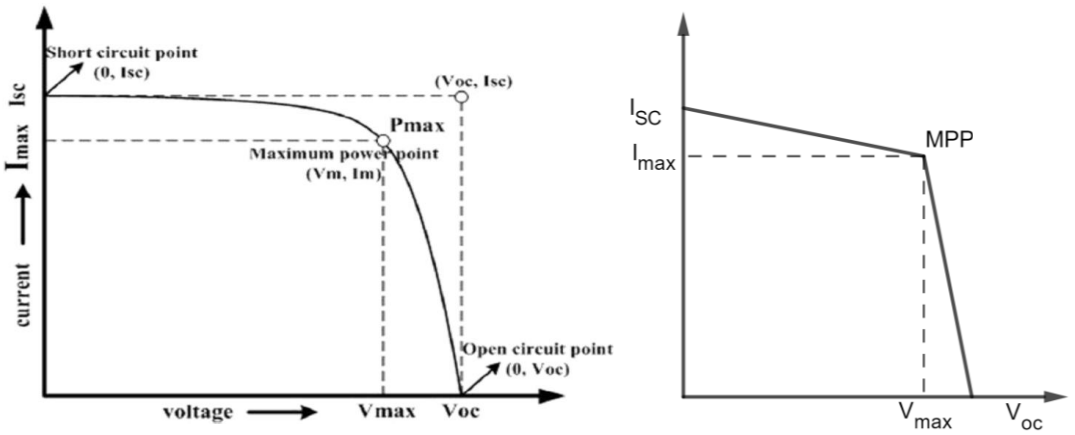
To complete the explanation of the framework, each of the modules consists of the main class, a helper that is used to install the module into a node, and containers that store every module installed to a node (each node have associated one container for Energy Source modules, another for Energy Harvester modules and another one for Device Models).

[13] ns-3 Documentation

3.1.2. Solar cells

The single diode model presented at Section 2.2 is very complete, as it considers so many parameters that describe the behaviour of a solar cells. However, the objective of this module is implementing a generic model that allows implementing any kind of solar cell or solar cells array. Because of that, the single diode model has been discarded and a custom model (which is explained through this section) has been implemented.

Regarding the V-I characteristic of a solar cell at Figure 8(a), it can be approximated into Figure 8(b).



(a) Real solar cell V-I characteristic

(b) Proposed solar cell V-I characteristic

Figure 8: Real vs proposed models for solar cells

As it can be seen, the proposed design simplifies the I-V characteristic in a way that there is no need of a high demand of resources to perform the calculation, which leads to a reduced simulation time. So, the behaviour of the solar cell is described with a non-linear function, which consists of two lines with a discontinuity point at Maximum Power Point (MPP). The equation that describes those lines is:

$$I = \begin{cases} \dfrac{I_{MAX} - I_{SC}}{V_{MAX}} \cdot V + I_{SC} & for\ \ 0 < V < V_{MAX} \\ \dfrac{-I_{MAX}}{V_{OC} - V_{MAX}} \cdot (V - V_{MAX}) + I_{MAX} & for\ \ V_{MAX} < V < V_{OC} \end{cases} \quad \textbf{(11)}$$

Besides the need of implementing this function, there are also various approaches to design the integration of this module to DSS-SIM. One approach could be done condidering the solar cells as a DSS-SIM Model. This approach considers the solar as a DSS-SIM

Model object, what accomplishes the operation principle of the simulator. Despite that, another approach can also be taken in consideration. As there already exists a native NS-3 implementation for energy harvesters, this class can be inherited and aggregate the solar into module into the solar cells into the Energy Framework environment. Another consideration to take into account is that an interface that adapts Energy Framework to DSS-SIM will be performed at Section 3.1.4, so there is no need to adapt twice the model.

Finally, and as consequence of the exposed information, finally the next designed has been adopted:

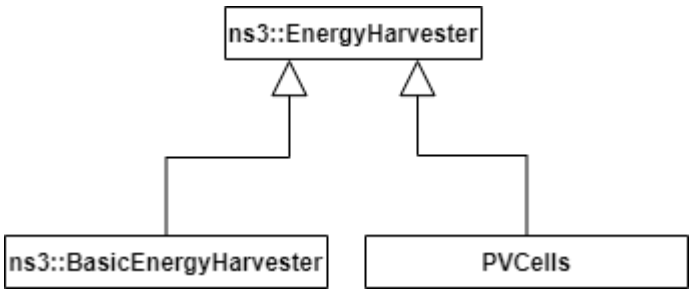


Figure 9: *ns3::EnergyHarvester* module with implemented solar cells

The results for the chosen design for modelling a solar cell will be exposed at Section 4.1.1

3.1.3. Battery

With all the presented battery models at Section 2.3, we need to build up a table to evaluate all the requirements described at Section A.1.1. Table [] presents the requirements related to battery model, reviewing if each requirement is accomplished for the proposed models. This will allow us choosing a model based on which is more suitable for our case of study. In addition to the requirements, in the table will also appear if the model is already implemented in a NS-3 library.

Requirement ID	Shepherd model	Copetti model	MSM	eMSM
Suitability + single equation	X		X	
Easy obtainable parameters	X	X	X	X
Overcharging protection				
NS-3 implementation			X	

Figure 10: Comparison between proposed battery models

Looking at the table, the model that best suits our case of study is the MSM. The main characteristics of this model are that the charge/discharge equation is the same for both states, and the only difference is the current direction, as it has been explained in Section 2.3.3. Another important characteristic is that it is already implemented in the NS-3 energy frame, which simplifies the project as there is no need of re-implementing the class again.

3.1.4. Energy subsystem interface

Once the NS-3 energy framework has been presented in Section 3.1.1, the designed solar cells in Section 3.1.2, and the chosen battery model in Section 3.1.3, it is time to present how they interact with each other and with the DSS-SIM.

As there is no relation between a NS-3 model and the DSS-SIM, an interface shall be designed to include the energy framework from NS-3 into the DSS-SIM. This interface consists of three small interfaces, each one for each of the modules implemented in the energy framework. Figure 11 shows the designed interface:

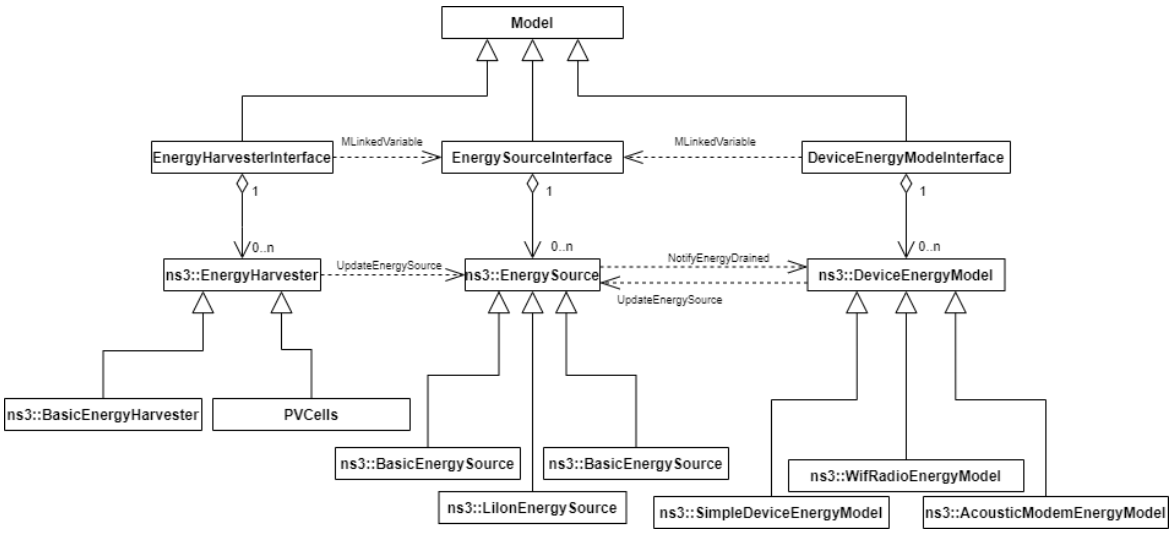


Figure 11: UML diagram for designed EPS

As it can be seen, the interface preserves the operation of the energy framework as in [23], [24] E*nergySourceInterface* has four input variables, two from *EnergyHarvesterInterface* and two more from *DeviceEnergyModelInterface*, represent the provided and consumed power and current from the harvesters and the devices. In a real application, the energy source would have as input the harvested current, as it charges the source, and would have as output the consumed energy, as it discharges the source. The designed application does not work this way, as the energy source needs to know the consumption of the model in order to calculate the remaining energy stored. Because of either the harvesting current/power and the consuming current/power can change during a simulation, the variables have been defined as inputs, so *EnergySourceInterface* is able to detect changes on them.

This designed interface allows to interconnect Model created objects directly into the interface. Despite that, and as all the EPS modules are inherited from NS-3, the final implementation of the interface relies on EnergySource::UpdateEnergySource method, which already retrieves the providers and consumers from the NS-3 containers where they are stored. Because of that, functionality of the linked variables between interfaces is just notifying the *EnergySourceInterface* when a current changes, so the interface can call EnergySource::UpdateEnergySource method.

The obtained results for this approach are shown in Section 4.1.2.

3.2. Satellite Contact Link Emulator

As it has been mentioned, this project has been developed from the ground. The first approach done is presented in Figure 12.

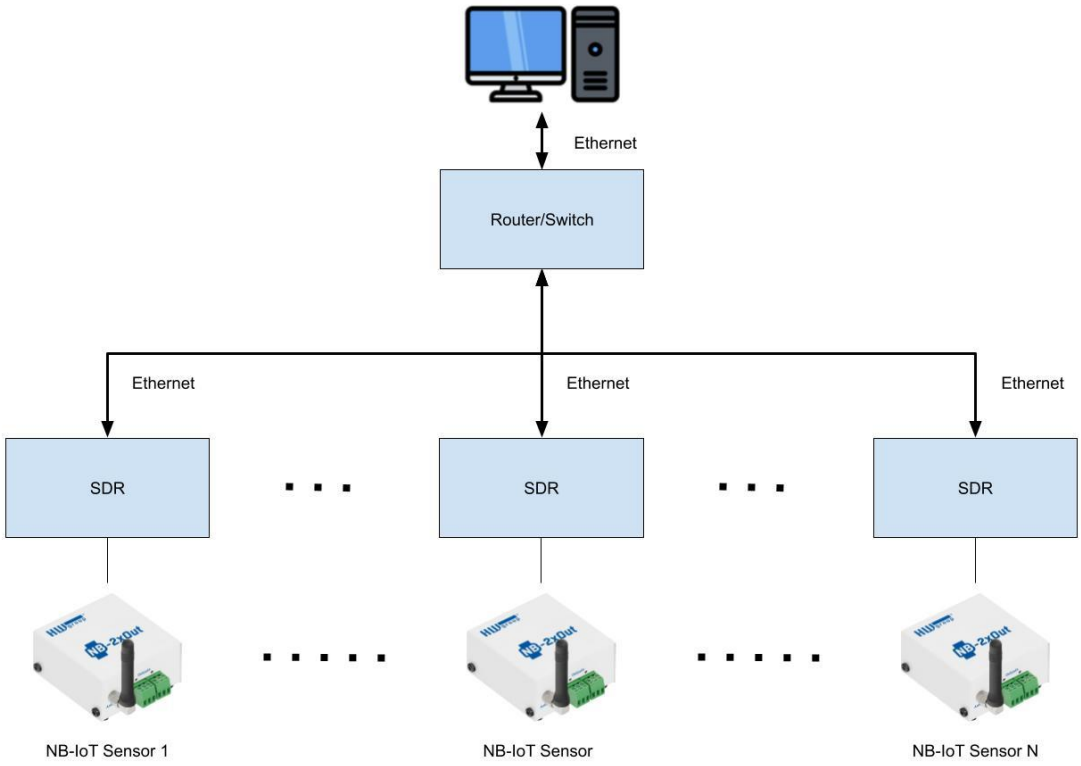


Figure 12: SCE initial scheme

The main idea of this approach can be split into two main modules: the central PC and the SDRs. The functionality of the central PC is detecting all the devices connected into the network and propagating its orbits. Once the orbits are propagated, there is the need to compute if there is a contact between the nodes, and then determine the attenuation, Doppler shift, and delay between them. The operation of propagating the orbit and the calculation of the parameters must be done every time step (to be determined), in order to emulate the contacts “continuously”. Once all these parameters are calculated, this computed matrix is passed via Ethernet to the SDRs. They receive this matrix and use it to compute the existing communications channel between the other nodes. Once built the communications channel, SDR can use the network to exchange data, which must be constantly converted from analog to digital and vice versa. As it has been said, this thesis would focus on the development of the first module, while [5] will focus on the development of the second module.

3.2.1. Orbit propagation

In Section 2.5, the current orbital models have been described. Among all of them, the most complete ones are SGP4 and SGP8, as they take into consideration more phenomena that affect the orbit trajectory. The most widely used is the SGP4, and the main difference between SGP4 and SGP8 is the re-entry calculation (which is not relevant for SCE), SGP4 will be the one implemented in this project. To implement its equations, we have searched for already implemented SGP4 libraries or software. An interesting library is Astrodynamics Software[14], which is a software developed by Vallado using its SGP4 propagator described in [27], [28]. This library is coded in various programming languages, such as C++. Despite the freedom to choose the programming language, there is not much documentation, and functions are defined at a very low-level.

Another available software which implements that orbit propagator is Satellite Toolbox[15]. This library is programmed in Julia, which is a high-level dynamic programming language. The toolbox allows the simulation of different orbitals models such as J2, J4, and SGP4, but its high-level code makes it simpler to use. Another feature to consider is the performance of Julia as programming language, which is high despite being a dynamic programming language [29]. Reviewing the toolbox API, an orbit can be propagated by three functions:

- *propagate!*: The orbit will be propagated by t (s) from the orbit epoch, which is defined in the initialization. This function returns a tuple with three values, but only two are of our interest: the position vector, and the velocity vector represented in the inertial reference frame.
- *propagate_to_epoch!*: The input argument is an epoch (Julian Day) to which the orbit will be propagated. The returned elements are the same as in first method,
- *step!*: The orbit is propagated by Δt (s) from the last propagation instant. This function returns the same information as in the first described method.

Analyzed all the orbit determination functions, the last one is the most suitable for our use-case, as determining the obit will be constantly done. Before the call to these functions, an orbit propagator must be defined, and depending on the choice some other inputs will be required. For our use case, the SGP4 will be the propagator, so it requires the Two-Line elements (TLE) as inputs.

The election of this toolbox to achieve the orbit determination influences the whole project, as it will be carried out in Julia programming language. Despite the advantages of Julia, it has also some limitations associated, mainly related to the lack of maintained libraries.

3.2.2. ISL channels effects

The main goal of the project is emulating a ISL communication channel. To achieve that, it will be modelled as functions of three parameters: attenuation, delay, and Doppler shift.

[14] CelesTrak: Astrodynamics Software by David Vallado
[15] Satellite Toolbox

As we want to emulate an ISL channel, the propagation medium is free space, so Free Space Path Losses (FSPL) will be modelled [30]:

$$FSPL = \left(\frac{4\pi d}{\lambda}\right)^2 = \left(\frac{4\pi df}{c}\right)^2 \quad \textbf{(12)}$$

where:

- d = Distance between satellites
- λ = Signal wavelength
- f = Signal frequency
- c = Speed of Light

Another formulation of equation **(12)** is expressing the result in decibels:

$$FSPL\ (dB) = 10 \cdot \log_{10}\left(\frac{4\pi df}{c}\right)^2 = 20\log_{10}(d) + 20\log_{10}(f) + 20\log_{10}\left(\frac{4\pi}{c}\right) \quad \textbf{(13)}$$

The propagation delay uses this simple equation, considering the speed of light of speed as propagation speed:

$$t = \frac{d}{c} \quad \textbf{(14)}$$

Finally, modelling the Doppler shift is done as explained at [31]:

$$\Delta f_1 = \frac{f_c}{c} \cdot (|\overrightarrow{v_1}| \cdot \cos(\vartheta_1) - |\overrightarrow{v_2}| \cdot \cos(\vartheta_2)) \quad \textbf{(15)}$$

where:

- Δf_1 = Shift observed by device 1
- f_c = Carrier frequency
- $\overrightarrow{v_1}$ = Velocity vector of device 1
- $\overrightarrow{v_2}$ = Velocity vector of device 2
- ϑ_1 = Angle formed by velocity of device 1 and $\overrightarrow{r_2} - \overrightarrow{r_1}$
- ϑ_2 = Angle formed by velocity of device 2 and $\overrightarrow{r_2} - \overrightarrow{r_1}$

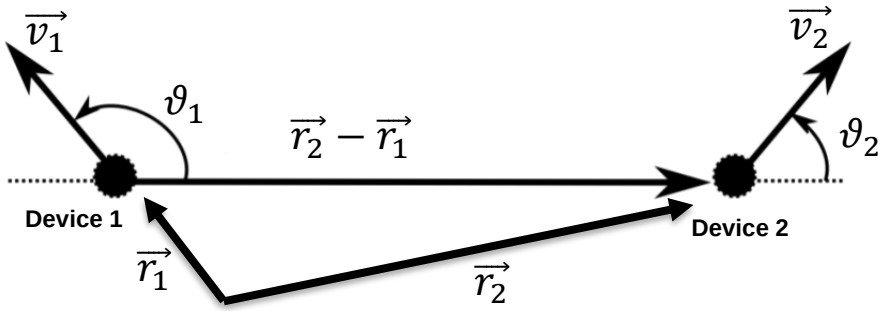
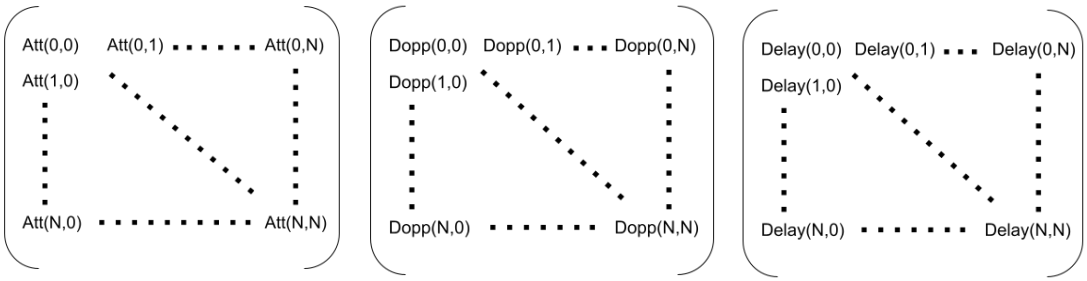


Figure 13: Doppler shift parameters relation

The implementation of this function has been done in Julia from scratch, as there are no libraries that computes them. The validation of the implemented functions is done in Section 4.2.1.

3.2.3. Control message structure

Now the way the channel parameters are obtained has been explained, is needed an explanation of how these parameters are serialized into messages. The matrix formed by all these parameters is presented next:



(a) Attenuation matrix (b) Doppler matrix (c) Delay matrix

Figure 14: Parameters matrix

Rows of the matrix represent the receiver SDR parameters, while columns represent the transmitters For example, row 1 contains all the parameters to emulate the communication channel when SDR1 receives data. However, matrices are still not the final message strcture.

To control this message strcuture, some serialization formats have been implemented, as it figures out at requirements (Annex A.2.1). During the implemenatation of the serializers, it was found out that its use was trivial, so finally embedded serializers like JSON or MsgPck have finally not been implemented. Because of that, a custom architecture to control messages has been adpoted. Basically, the implemented architecture consists on retrieving each row of the matrices, following the order Attenuation->Doppler->Delay, and putting them following the other inside a vector. The resulting vector results in:

Att(0,:)	Att(1,:)	. . .	Att(N,:)	Dopp(0,:)	. . .	Dopp(N,:)	Delay(0,:)	. . .	Delay(N,:)

Figure 15: Custom message structure

3.2.4. Integration of central PC via Ethernet

After reviewing how the orbits determination is done, and how the parameters are obtained and converted into messages, it is time to discuss how the PC will be integrated over Ethernet. PC only sends control messages, but it never intervenes on the data flow between the SDRs. To be sure that control messages do not intervene on data messages, another design was proposed as it can be seen in Figure 16. That design relied on that SDRs allowed two digital inputs, but the selected devices only have one, what made discarding that design.

The connection between SDRs and the central PC is performed by sockets. The design structure to control messages sets the PC as a server, and all the devices as clients. To accomplish this approach, the SDRs must implement multithreading, as one thread must be always listening for control data coming the central PC, because the parameters matrix is constantly being updated and sent to the SDRs.

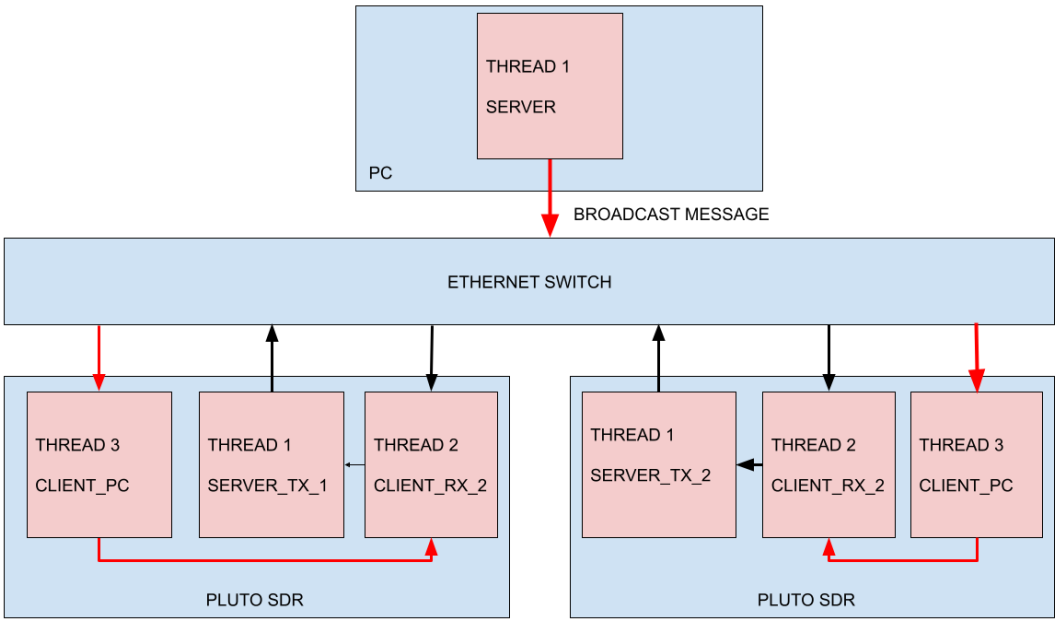
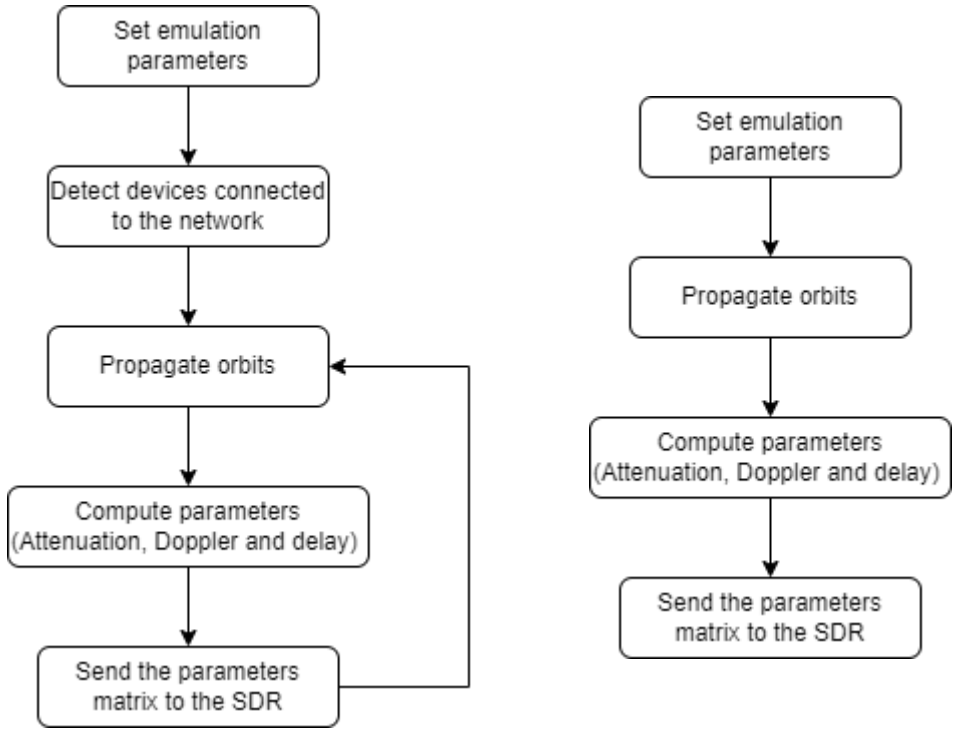


Figure 16: Sockets structure with multithreading

But the SDRs have a limited performance, which is even more limited when using just the processor and not the FPGA, as it is explained at [5]. Because of that limitation, multithreading will not be finally implemented, and that change of structure has a major impact on the functionality of the system. The original and final workflow of the central PC are shown next:



(a) Original central PC workflow (b) Final central PC workflow

Figure 17: Central PC workflows

As it can be seen, the original design included a continuous loop where orbit determination and parameter calculations are performed. However, due to the problems associated to multithreading, this approach was discarded, and a simpler design has been finally implemented: it does not include the loop, and it only determines the orbits and the parameters once. This is not a real SCE, but this design demonstrates the feasibility of the project.

4. Tests and results

This section presents the tests and the obtained results for the implement designs. The section is divided in two parts, Section 4.1 presents the performed test to validate the DSS-SIM implementation, while Section 4.2 presents the validation methods for SCE.

4.1. DSS-SIM

To consider that implementation of a module is correct, unit tests for each implemented module must be performed. This kind of tests verifies every function of the module. These tests are done using Google Test (GTest)[16].

4.1.1. Solar cells

Due to the complexity of current existing solar cells models, the final implemented model has been designed especially for obtaining an approximation of the I-V characteristics. To validate the design, two main tests have been performed:

1. A custom dataset of expected results has been built in order to check the function that has been implemented. This dataset has been constructed using GeoGebra[17], by graphically describing the implemented function and obtaining a set of outputs for some input points. This test was passed successfully.

2. Furthermore, another test has been performed to check the accuracy of the implemented model. This has been done by comparing an already solar cell module in MATLAB with the implemented. The results are shown in Figure 19, which represents the curve fitting, and in Figure 18, which represents the error produced by the model. As it can be seen, the error is not very high (< 0,5 A) before MPP, but after that point, the error is considerable, as it may be up to 2,25 A.

[16] GoogleTest
[17] Calculadora gráfica - GeoGebra

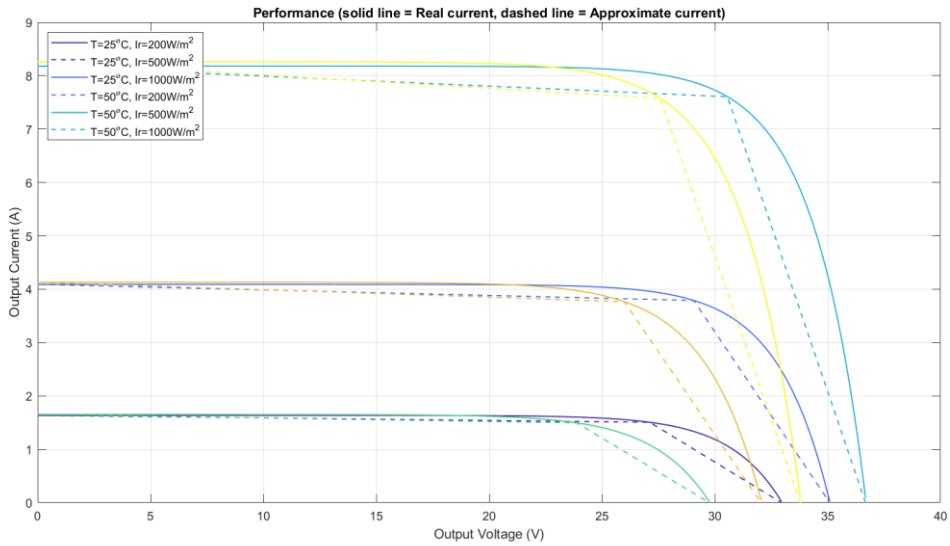


Figure 18: Solar cells fitting into real I-V curves

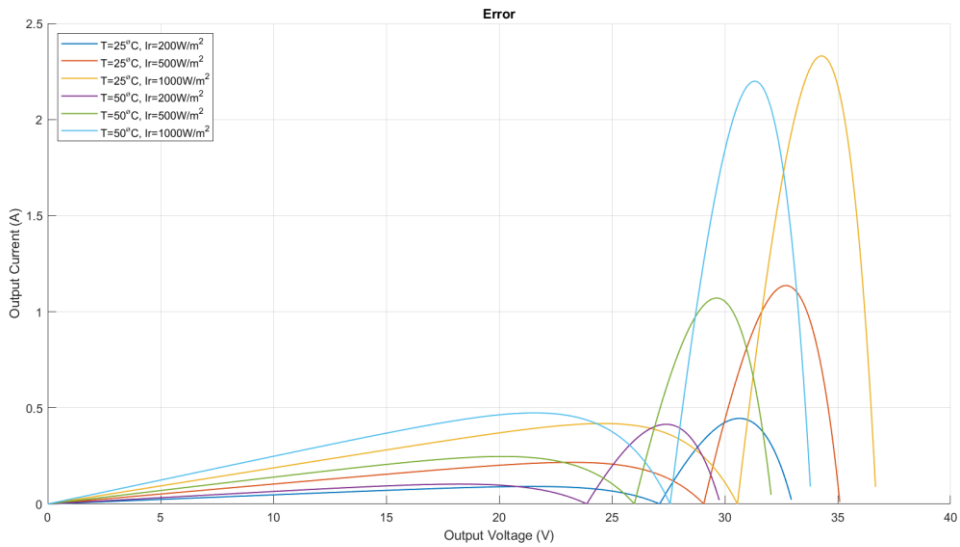


Figure 19: Solar cells model error

4.1.2. Energy subsystem interface

In order to validate the designed interface, two tests have been performed:

1. Linking each interface input/output variables to check if the links are constructed correctly. Unfortunately, this test has not been passed because the variables are not linked, which makes useless the designed interfaces. However, this does not mean that the interface structure is wrong, it just means that the creation of the variables is done wrong.

2. A set of simulation scenarios (i.e., solar cell as harvester with a battery as energy source) have been defined in order to test the designed interface structure. However, this test has not been performed, as it depends on the first performed. So, the interface structure has not been validated by any method.

Probably with some more time the linking issues would have been solved, allowing to test the implemented interface structure.

4.2. Satellite Contact Link Emulator

Three main tests have been performed to ensure that the designed central PC works correctly, which consist of:

1. Test the orbit propagation and the parameters calculation
2. Test the sending of parameters matrix to the local host
3. Test the integration of sockets, and a continuous orbit propagation and parameter calculation into the SDRs.

4.2.1. Orbit determination and channel effects calculations

The first test to be performed has been the determination of the orbit, as well as the calculations of attenuation, Doppler shift, and delay. To perform that, a set of TLEs are obtained from Celestrak[18], as they are needed to initiate the orbit propagator. The used TLEs to perform this test can be found at Annex[]. The other parameters that define this scenario are: $f_c = 860\ MHz$ and $N_{devices} = 4$. The outputs matrices of the performed tests are:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 8.380\ km & 8.380\ km & 8.590\ km \\ 8.380\ km & 0 & 673{,}52\ km & 678{,}74\ km \\ 8.380\ km & 673{,}52\ km & 0 & 395{,}17\ km \\ 8.590\ km & 678{,}74\ km & 395{,}17\ km & 0 \end{pmatrix}$$

(a) Contact matrix (b) Distance matrix

Figure 20: Contact and attenuation matrices

[18] CelesTrak

0	0	0	0
0	0	147,71 dB	147,77 dB
0	147,71 dB	0	143,01 dB
0	147,77 dB	143,01 dB	0

(a) Attenuation matrix

0	0	0	0
0	0	3321,34 Hz	23073,90 Hz
0	3321,34 Hz	0	7098,51 Hz
0	23073,90 Hz	7098,51 Hz	0

(b) Doppler matrix

0	0	0	0
0	0	2,25 ms	2,26 ms
0	2,25 ms	0	1,32 ms
0	2,26 ms	1,32 ms	0

(c) Delay matrix

Figure 21: Attenuation, Doppler, and delay matrices

The contact and the distances matrices are included into the results to verify the correct functioning of the test, as if there is no contact (a '0' in Figure 20(a)), none of the parameters are calculated (only distances for verifying purposes). This also occurs for self-contact positions, which are the values stored at the main diagonal. The distance matrix, Figure 20(b), is used to validate the calculation of attenuation.

The main parameters to validate are Doppler and attenuation, as the equations that describe them are more complex, and because the orbit determination is done by a reliable toolbox, so no validation is needed. Figure 21(a) shows the obtained attenuation matrix. As the FSPL model is used to compute its parameters, MATLAB in-built function *fspl()*[19] has been used to generate some graphics and compare them with obtained results:

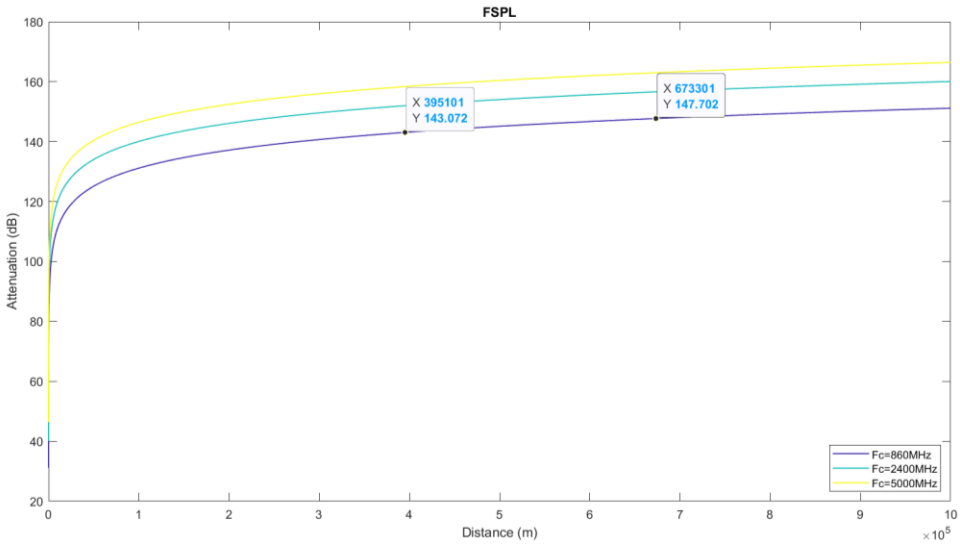


Figure 22: FSPL MATLAB model

[19] FSPL-MATLAB

The attenuation obtained in the MATLAB model is the same as the obtained from the attenuation matrix, so it can be said that the implementation of the attenuation model has been successful.

Figure 21(b) shows the Doppler shift obtained with the implemented model. Unfortunately, a proper data set or in-built function has not been found to validate these results. However, Table 1 at [31] presents some Doppler shifts for 600 km distance. Based on that table, the obtained Doppler shift have the same order as the presented there. So, besides not making a validation 100% accurate, it can be said that the implemented Doppler function does not have a high deviation.

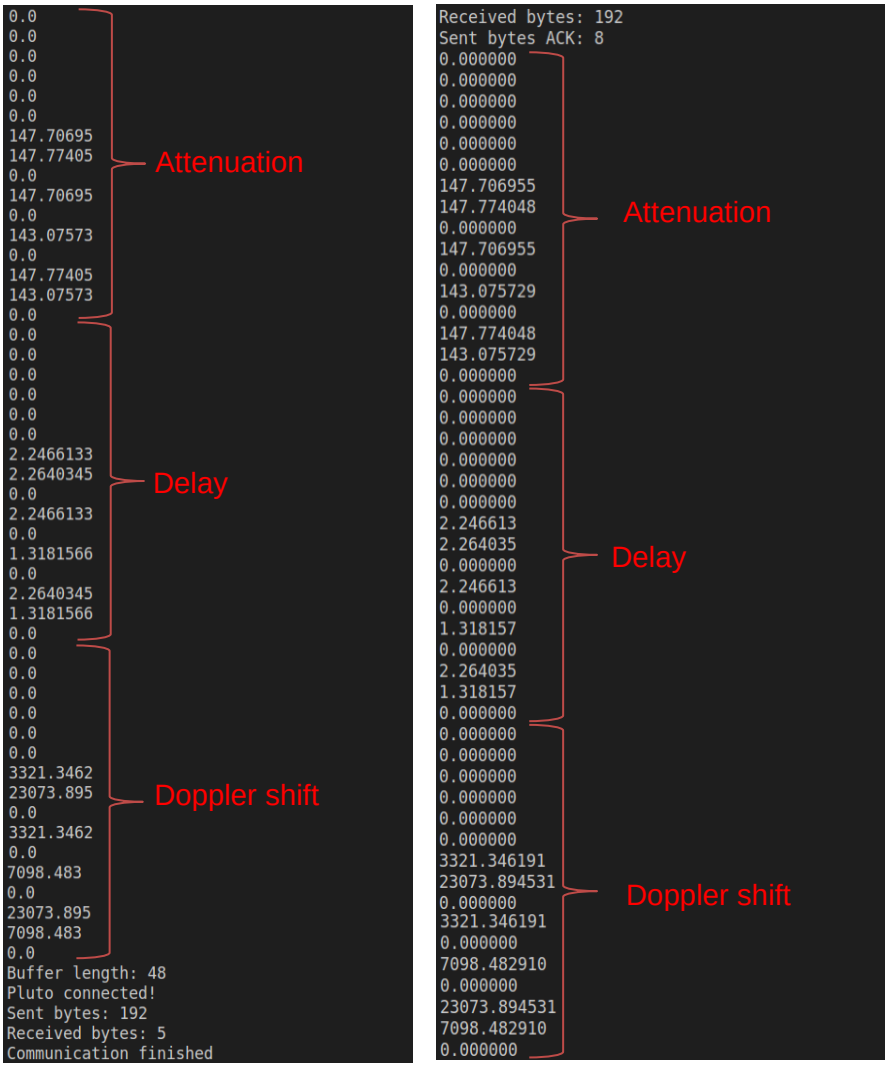
Finally, Figure 21: Attenuation, Doppler, and delay matricesFigure 21(c) presents the obtained delay matrix. The delays are less than the delay produced by the network, so this effect can not be represented with the designed architecture.

By reviewing the test carried out, it can be said that the determination of the orbit and the calculation of the parameters are done properly.

4.2.2. Transmission of the channel effects parameters via Ethernet

The second performed test has been the implementation of a server-client architecture to transmit the computed parameters to the SDRs. However, before implementing the architecture in the SDR, the chosen sockets have been tested in order to ensure its correct functioning. This test consists in the implementation of a server coded in Julia, that also performs the orbit propagation and the calculation of the parameters, and sends them to an implemented client in C (which is the programming language chosen for SDRs code implementation [5]). The server is run at the PC, while the client runs at local host. When the client receives the message, send an ACK to confirm the reception.

The emulation scenario is the same as the previous test, so the send parameters will be the same presented at Figure 20. The results are:



(a) Server side (b) Client side

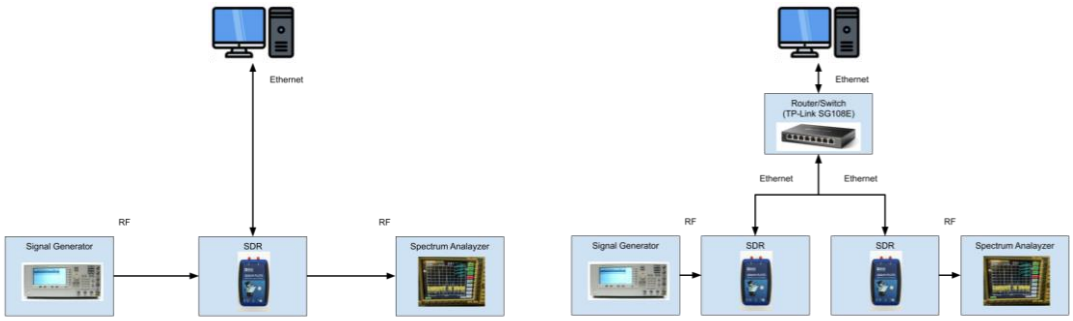
Figure 23: Server-Client parameters transmission

The matrix parameters are serialized as explained in Section 3.2.4. Parameters are correctly transmitted with no error, as it can be checked by comparing transmitted and received bytes. Both are 192 bytes, which correspond to the number of elements ($3 \cdot N^2 = 48$ in this scenario with 4 devices), multiplied by the length of the variable which is used to store the parameters. In this case, the parameters are stored in *floats*, whose length is 4 bytes. Moreover, the correct functioning can also be checked by the transmission of the ACK by the client.

During this test, the external tools ZeroMQ and MsgPack tried to be implement. However, it was not possible to integrate the tools, so their implementation was finally discarded due to lack of time.

4.2.3. Software integration on SDRs

Last performed test has been the continuous transmission of the parameters over the SDRs. It is the same test as the performed in Section 4.2.3, but the client runs at the SDRs instead of local host. Two different architectures to perform the test have been implemented:



(a) 1 SDR testbed

(b) 2 SDR testbed

Figure 24: Final testbeds

From the point of view of central PC, both testbeds are the same, because in Figure 24(b) only one of the SDRs receive the transmitted parameters. The results of this test have been that the connection between central PC and one SDR had been achieved. Despite that, the test can be considered as unsuccessful because the continuous retransmission of the parameters has not been achieved. The reason not achieving that operation is related to multithreading issues explained during Section 3.2.4

5. Conclusions and future development:

This final degree thesis aimed to contribute to ISL simulation by extending the DSS-SIM capabilities, by creating an EPS model that included battery models and solar cells model. Moreover, it also aimed to develop an SCE due to the lack of cost-effective alternatives. However, not all the proposed objectives have been met.

Focusing on the development of the EPS, it was decided to adapt the NS-3 Energy Framework to the simulation engine instead of designing an own EPS from scratch, as most functionalities of the EPS were already implemented on it. This decision also considered the NS-3 community, as the functionalities of the framework would probably be extended by the in the future. However, the chosen framework does not support solar cells, so they had to be implemented. The chosen model is a simplification of the real behaviour of solar cells, but it is enough to represent the impact of ISL on an EPS.

Regarding the DSS-SIM list of objectives in Section 1.1, two of them have been achieved, while the other one has not. The achieved objectives have been the implementation of solar cells (1), which have been validated by testing, and the implementation of a battery (2), which has been implemented by adapting the NS-3 Energy Framework. However, the integration of both modules into an EPS (3) has been implemented, but its validation failed; so, it can be considered as partially achieved. Probably, the validation would have been done if there were more time to develop the project.

The future development of this project will be related to the validation of the designed interface, and then design the descoped modules presented in Annex A.1.1, which are a memory model and an Earth traffic model.

Reviewing the development done in terms of the SCE, the project was divided into three small testbeds to gradually extend its functionalities. At the first testbed, the calculation of the channel effects was performed. At the second one, the transmission of the channel effects parameters to a localhost was performed. At the final testbed, these computed parameters were transmitted to the SDR.

The defined objective of the SCE was the "*creation of a prototype that allows communications between two end-devices, connected into a network composed by Software Defined Radios (SDR) and a central PC that performs the ISL emulation"*. It can be said that it has been achieved, as a basic prototype has been proved in Section 4.2.3. However, this is a limited prototype, as no continuous transmission is performed by the central PC.

The future development is related to achieving this continuous retransmission, which would be the operation of a real SCE.

Bibliography:

[1] C. Araguz, E. Bou-Balust, and E. Alarcón, “Applying autonomy to distributed satellite systems: Trends, challenges, and future prospects”, doi: 10.1002/sys.21428.

[2] J. A. R. de Azua, A. Calveras, and A. Camps, “Internet of Satellites (IoSat): Analysis of Network Models and Routing Protocol Requirements,” *IEEE Access*, vol. 6, pp. 20390–20411, Apr. 2018, doi: 10.1109/ACCESS.2018.2823983.

[3] J. A. Ruiz de Azúa, “Contribution to the development of autonomous satellite communications networks : the internet of satellites,” Universitat Politècnica de Catalunya, 2020. [Online]. Available: http://hdl.handle.net/2117/346653

[4] J. A. Ruiz-De-Azúa, C. Araguz, A. Calveras, E. Alarcón, and A. Camps, “Towards an integral model-based simulator for autonomous earth observation satellite networks,” *International Geoscience and Remote Sensing Symposium (IGARSS)*, vol. 2018-July, pp. 7403–7406, Oct. 2018, doi: 10.1109/IGARSS.2018.8517811.

[5] A. Dolz Puig, “Contributions to simulation and emulation of Inter-Satellite Links,” Universitat Politècnica de Catalunya, Barcelona, 2022.

[6] B. Barritt, K. Bhasin, W. Eddy, and S. Matthews, “Unified approach to modeling & simulation of space communication networks and systems: Integrating network stack and astronautical physics simulators,” *2010 IEEE International Systems Conference Proceedings, SysCon 2010*, pp. 133–136, 2010, doi: 10.1109/SYSTEMS.2010.5482493.

[7] J. Puttonen, B. Herman, S. Rantanen, F. Laakso, and J. Kurjenniemi, “Satellite Network Simulator 3 Workshop on Simulation for European Space Programmes (SESP)”.

[8] H. ~J. Hovel, “Solar cells,” *NASA STI/Recon Technical Report A*, vol. 76, p. 20650, Jan. 1975.

[9] R. Abbassi, A. Abbassi, M. Jemli, and S. Chebbi, “Identification of unknown parameters of solar cell models: A comprehensive overview of available approaches,” *Renewable and Sustainable Energy Reviews*, vol. 90, pp. 453–474, Jul. 2018, doi: 10.1016/J.RSER.2018.03.011.

[10] J. A. Gow, “Development of a model for photovoltaic arrays suitable for use in simulation studies of solar energy conversion systems,” pp. 69–74, Nov. 2005, doi: 10.1049/CP:19960890.

[11] F. Ghani, G. Rosengarten, M. Duke, and J. K. Carson, “The numerical calculation of single-diode solar-cell modelling parameters,” *Renewable Energy*, vol. 72, pp. 105–112, Dec. 2014, doi: 10.1016/J.RENENE.2014.06.035.

[12] J. Meng, G. Luo, M. Ricco, M. Swierczynski, D. I. Stroe, and R. Teodorescu, “Overview of Lithium-Ion Battery Modeling Methods for State-of-Charge Estimation in Electrical Vehicles,” *Applied Sciences 2018, Vol. 8, Page 659*, vol. 8, no. 5, p. 659, Apr. 2018, doi: 10.3390/APP8050659.

[13] C. M. Shepherd, “Design of Primary and Secondary Cells,” *Journal of The Electrochemical Society*, vol. 112, no. 7, p. 657, 1965, doi: 10.1149/1.2423659.

[14] O. Tremblay, L. A. Dessaint, and A. I. Dekkiche, “A generic battery model for the dynamic simulation of hybrid electric vehicles,” *VPPC 2007 - Proceedings of the 2007 IEEE Vehicle Power and Propulsion Conference*, pp. 284–289, 2007, doi: 10.1109/VPPC.2007.4544139.

[15] J. B. Copetti, E. Lorenzo, and F. Chenlo, “A general battery model for PV system simulation,” *Progress in Photovoltaics: Research and Applications*, vol. 1, no. 4, pp. 283–292, Oct. 1993, doi: 10.1002/PIP.4670010405.

[16] O. Tremblay and L. A. Dessaint, “Experimental Validation of a Battery Dynamic Model for EV Applications,” *World Electric Vehicle Journal 2009, Vol. 3, Pages 289-298*, vol. 3, no. 2, pp. 289–298, Jun. 2009, doi: 10.3390/WEVJ3020289.

[17] H. C. Bui and L. Franck, “Cost effective emulation of geostationary satellite channels by means of software-defined radio,” *2014 IEEE International Workshop on Metrology for Aerospace, MetroAeroSpace 2014 - Proceedings*, pp. 538–542, 2014, doi: 10.1109/METROAEROSPACE.2014.6865984.

[18] C. Trivedi Harsh, S. Gandhi, and R. K. Singh, “Design and Development of Dynamic Satellite Link Emulator with Experimental Validation,” *2021 12th International Conference on Computing Communication and Networking Technologies, ICCCNT 2021*, 2021, doi: 10.1109/ICCCNT51525.2021.9579962.

[19] R. Murawski, K. Bhasin, D. Bittner, A. Sweet, R. Coulter, and D. Schwab, “Hardware and Software Integration to Support Real-Time Space-Link Emulation”.

[20] D. J. Fonte, J. Orbital Analyst, C. Sabol Orbital Analyst, D. A. Danielson, and M. W. R Dyar, “Comparison of Orbit Propagators in the Research and Development Goddard Trajectory Determination System (R & D GTDS). Part I: Simulated Data,” 1995.

[21] F. R. Hoots and R. L. Roehrich, “Models for Propagation of NORAD Element Sets,” Dec. 1980.

[22] C. Araguz López, “In pursuit of autonomous distributed satellite systems,” Universitat Politècnica de Catalunya, 2019. Accessed: Jun. 13, 2022. [Online]. Available: http://hdl.handle.net/2117/175253

[23] H. Wu, S. Nabar, and R. Poovendran, “An Energy Framework for the Network Simulator 3 (ns-3),” 2011.

[24] C. Tapparello, H. Ayatollahi, and W. Heinzelman, “Extending the Energy Framework for Network Simulator 3 (ns-3),” Jun. 2014, doi: 10.1145/2675683.2675685.

[25] D. N. Rakhmatov and S. B. K. Vrudhula, “An analytical high-level battery model for use in energy management of portable electronic systems,” *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers*, pp. 488–493, 2001, doi: 10.1109/ICCAD.2001.968687.

[26] D. Rakhmatov, S. Vrudhula, and D. A. Wallach, “Battery lifetime prediction for energy-aware computing,” *Proceedings of the International Symposium on Low Power Electronics and Design, Digest of Technical Papers*, pp. 154–159, 2002, doi: 10.1109/LPE.2002.146729.

[27] D. A. Vallado and W. D. McClain, *Fundamentals of astrodynamics and applications*. Kluwer Academic Publishers, 2001. Accessed: Jun. 16, 2022. [Online]. Available: https://link.springer.com/book/9780792369035

[28] D. A. Vallado, P. Crawford, R. Hujsak, and T. S. Kelso, "Revisiting spacetrack report #3," *Collection of Technical Papers - AIAA/AAS Astrodynamics Specialist Conference, 2006*, vol. 3, pp. 1984–2071, 2006, doi: 10.2514/6.2006-6753.

[29] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A Fresh Approach to Numerical Computing," *http://dx.doi.org/10.1137/141000671*, vol. 59, no. 1, pp. 65–98, Feb. 2017, doi: 10.1137/141000671.

[30] J. G. Proakis and M. Salehi, *Digital Communications Fifth Edition*, 5th ed. McGraw-Hill, 2007. Accessed: Jun. 16, 2022. [Online]. Available: www.mhhe.com

[31] P. Pedrosa, D. Castanheira, A. Silva, R. DInis, and A. Gameiro, "A State-Space Approach for Tracking Doppler Shifts in Radio Inter-Satellite Links," *IEEE Access*, vol. 9, pp. 102378–102386, 2021, doi: 10.1109/ACCESS.2021.3098562.

[32] J. R. Spradley and W. L. Fang, "The role of electricity in sustainable development," *Natural Resources Forum*, vol. 21, no. 1, pp. 61–67, Feb. 1997, doi: 10.1111/J.1477-8947.1997.TB00673.X.

Appendices

A. Management

In this appendix an explanation of how the projects had been planned will be done. As both projects can be considered independent from each other, they have also been planned independently. Because of that, and as at the Gantt diagrams (Appendix A.1.3 and Appendix A.2.3) show, every working week has been dedicated to one of the projects, in order to achieve maximum performance in each one.

A.1. DSS-SIM

DSS-SIM project has been the most restricting one in terms of time. At the beginning of the project, a memory model (aiming to extend DSS-SIM capabilities in terms of physical modules), and a traffic model were proposed to implement. Despite reviewing the state of the art of each, and beginning the design of them, it was decided that they will be descoped. The reasons why this decision was made are explained at Appendix A.1.4. However, the non-implemented modules are still present on Gantt diagrams and Work packages, in order to demonstrate how the whole project was planned.

A.1.1. Requirements and specifications

Requirements define how the product will achieve the desired functionalities. The list of requirements for this project are presented at Table 1, including its type (functional or performance), and its validation method, which can be visual inspection of code (I), design revision (R), analysis (A) or test (T).

Table 1: DSS-SIM requirements

ID	Title	Type	Description	Verification Method [I/R/A/T]
BATTERY_010	Battery model	F	The chosen battery model shall be able to represent any type of battery technology and just using a single equation	T
BATTERY_020	Battery model inputs	F	The inputs of the chosen model shall be obtainable from the battery charge and discharge curves or from the manufacturer datasheet.	R
BATTERY_030	Overcharging	F	An additional protection method shall be implemented in order to avoid the overcharging of the battery if the chosen battery model does not take it into account.	T
BATTERY_040	Battery database	F	The output variable stored in the database shall be the variation of voltage during the simulation, the variation of inputs currents during the simulation	T

BATTERY_050	Battery discharged	F	A notification to all connected systems shall be done when the battery is completely discharged. When this happen, the systems connected to the battery must now that there is no more available energy, so they shall turn off.	T
BATTERY_060	Battery outputs	F	The battery shall provide voltage and current as outputs	T
ENERGY_070	Energy model	F	The energy model shall allow the user to create events of a change of the input current of the batteries whenever he/she desires.	T
ENERGY_080	Energy model consumption	F	It shall allow setting each state consumption or taking into account each module consumption	T
ENERGY_SOURCE_090	Solar cells models	F	The chosen battery model shall be able to represent any type of solar cells technology	T
ENERGY_SOURCE_100	Solar cells efficiency	F	The chosen model for solar cells shall take into account the efficiency of the cells. This efficiency will be set by the user.	T
ENERGY_SOURCE_110	Solar cell inputs	F	The input of solar cells shall be either come from the sun vectors or from an input light irradiance set by the user	T
ENERGY_SOURCE_120	Periodic irradiance	F	The user shall be able to set a period for the irradiance, in order to simulate eclipses.	T
ENERGY_SOURCE_130	Solar cells outputs	F	The output of the solar cells model shall be current	T
ENERGY_SOURCE_140	Basic energy source	F	It shall implement a basic energy source, where the input current will be fixed and chosen by the user	T
ENERGY_SOURCE_150	Energy source inputs	F	It shall be possible to choose between the basic model source or the solar cells	T

A.1.2. Work packages description

Once reviewed the state of the art of each module, and before starting the design, thesis advisors asked to plan how the project was going to be carried put. Because of that, the project was divided into different Work packages:

Project: DSS Simulator	WP ref.: WP1
Major constituent: Review of the state of the art	Sheet 1 of 5
Brief description: Review of current technologies used to achieve the implementation of the objectives modules.	Planned start date: 31/01/2022 Planned end date: 01/02/2022

	Start event: 31/01/2022 End event: 01/02/2022	
Internal task WP1.1: NS-3 + DSS-SIM: Review NS-3 API and DSS-SIM documentation to learn how the simulator works. Internal task WP1.2: Solar cell: Research for already implemented solutions to achieve the modelling of solar cells. Internal task WP1.3: Battery models: Review of existing battery models. Internal task WP1.4: Traffic model: Review of proposed models that describe the Earth traffic model.	Deliverables: -	Dates: -

Project: DSS Simulator	WP ref.: WP2	
Major constituent: Design	Sheet 2 of 5	
Brief description: Design the modules to implement.	Planned start date: 14/02/2022 Planned end date: 18/02/2022	
	Start event: 14/02/2022 End event: 18/02/2022	
Internal task WP2.1: Module description: Define the final functionalities that the module shall have. Internal task WP2.2: List of requirements: Define the requirements that the implemented modules shall have. Internal task WP2.3:	Deliverables: Design report which includes	Dates: 21/03/2022

Module high-level architecture: Define the final high-level architecture of the module Internal task WP2.4: UML diagram: Define how all the components of a module interacts with each other, Internal task WP2.5: Workflow diagram: Define a workflow diagram, where must appear all the functionalities that the modules must have.		

Project: DSS Simulator	WP ref.: WP3	
Major constituent: Implementation	Sheet 3 of 5	
Brief description: Implement the designed modules in WP2.	Planned start date: 14/02/2022 Planned end date: 18/02/2022	
	Start event: 14/02/2022 End event: 18/02/2022	
Internal task WP3.1: Solar cells: Implement the solar cells module, based on the designed made. Internal task WP3.2: Energy interface: Implement the energy interface module, based on the designed made. Internal task WP3.3: Memory model: Implement the memory module, based on the designed made. Internal task WP3.4: Solar cells: Implement the solar cells module, based on the designed made.	Deliverables: -	Dates: -

<table>
<tr><td>Project: DSS Simulator</td><td colspan="2">WP ref.: WP4</td></tr>
<tr><td>Major constituent: Testing</td><td colspan="2">Sheet 4 of 5</td></tr>
<tr><td rowspan="2">Brief description:
The models implemented at the previous Work package.</td><td colspan="2">Planned start date: 14/02/2022
Planned end date: 18/02/2022</td></tr>
<tr><td colspan="2">Start event: 14/02/2022
End event: 18/02/2022</td></tr>
<tr><td>Internal task WP4.1:
Solar cells: Test the solar cells implementation. Once tests are passed successfully, a revision of coding conventions shall be done before proceeding into pull-request submission.

Internal task WP4.2:
Energy interface: Test the energy interface implementation. Once tests are passed successfully, a revision of coding conventions shall be done before proceeding into pull-request submission.

Internal task WP4.3:
Memory model: Test the memory model implementation. Once tests are passed successfully, a revision of coding conventions shall be done before proceeding into pull-request submission.

Internal task WP4.4:
Traffic model: Test the traffic model implementation. Once tests are passed successfully, a revision of coding conventions shall be done before proceeding into pull-request submission.</td><td>Deliverables:
Pull-request of each module</td><td>Dates:
-</td></tr>
</table>

Project: DSS Simulator	WP ref.: WP5

<table>
<tr><td colspan="2">Major constituent: Documentation</td><td>Sheet 5 of 5</td></tr>
<tr><td colspan="2" rowspan="2">Brief description:
Generation of a final review which includes the design, testing, and implementation of the modules.</td><td>Planned start date: 14/02/2022
Planned end date: 18/02/2022</td></tr>
<tr><td>Start event: 14/02/2022
End event: 18/02/2022</td></tr>
<tr><td>Internal task WP5.1:
Generate documentation: Generation of a final review which includes the design, testing, and implementation of the modules.</td><td>Deliverables:
Final report</td><td>Dates:
21/06/2022</td></tr>
</table>

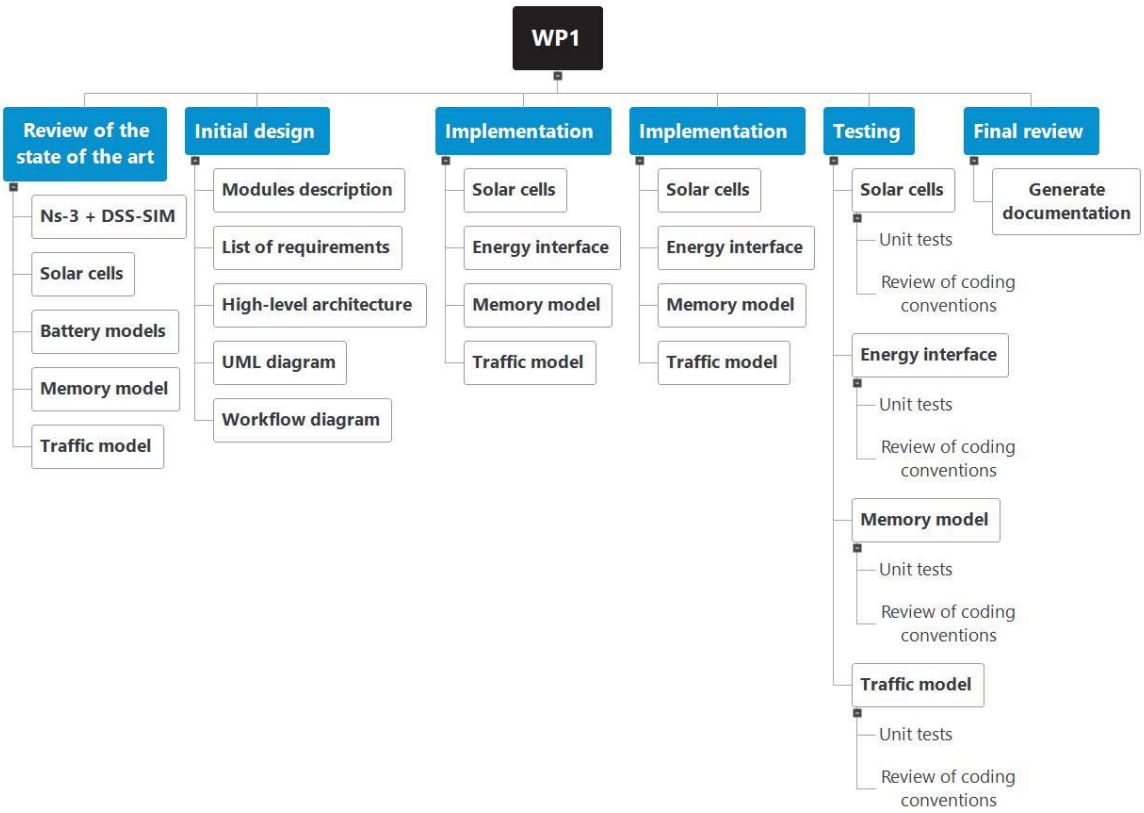


Figure 25: DSS-SIM work breakdown structure

A.1.3. Gantt diagrams

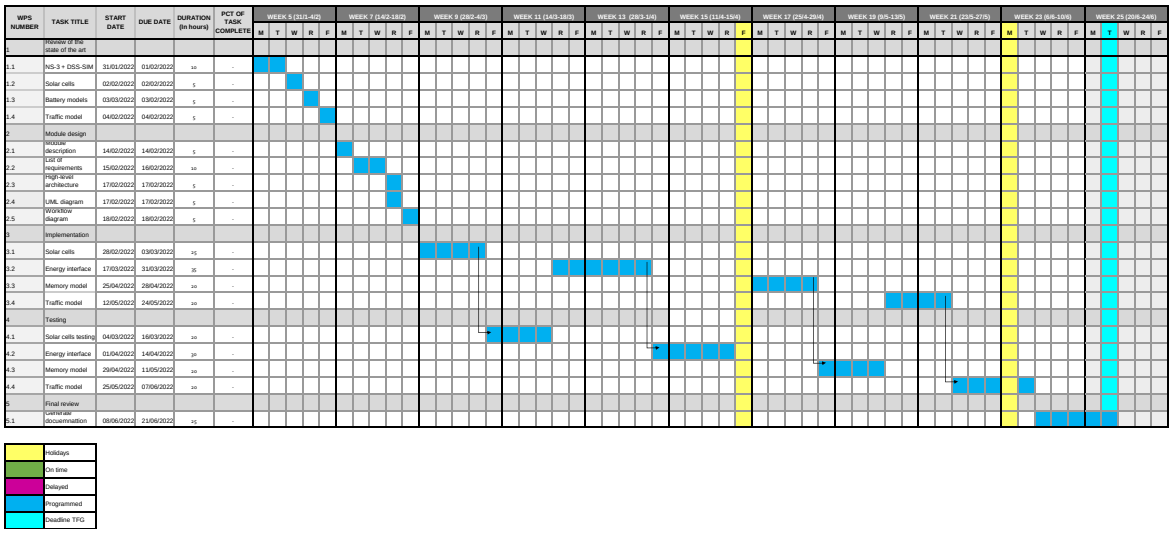


Figure 26: DSS-SIM initial Gantt diagram

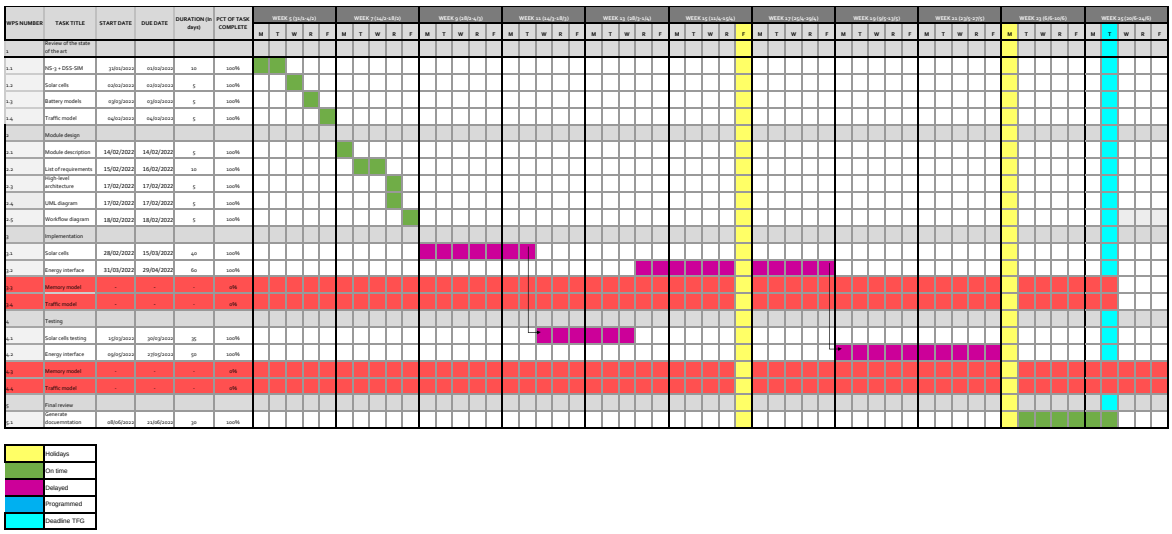


Figure 27: DSS-SIM final Gantt diagram

A.1.4. Deviations from initial proposal

As it has been said, more modules were planned to be implemented at the beginning of the thesis. However, and as it can be seen at Figure 27, they have not been finally implemented. The main reason for not achieving the original objectives has been a too optimistic estimation of the required time, as the complexity of those objectives was under-estimated during planning. This error can also be seen at the implemented module, as they not fulfil all the original objectives.

A.2. Satellite Contact Emulator

A.2.1. Requirements and specifications

The list of requirements for this project are presented at, including its type and its validation method.

Table 2: SCE requirements

ID	Name	Type	Description	Validation Method [I, R, A, T]
PC_010	Effects	F	It shall implement the following effects: Attenuation, Doppler shift and delay.	T
PC_020	Orbit propagation	F	It shall propagate the orbit following a SGP4 orbit propagators.	T
PC_030	Real-time calculations	F	It shall compute the desired channel effects in real-time.	T
PC_040	Mobility	F	It shall propagate the orbit / position of the hardware devices in real-time.	T
PC_050	Scenario Settings	F	It shall enable to configure the scenario to be emulated. (Orbit and channel settings)	T
PC_060	Device configuration	F	It shall be to configure devices as satellites or ground-stations	T
PC_070	Orbit parameters	F	It shall enable to assign an orbit or geographic coordinates to a connected hardware	T
PC_080	Earth rotation	F	It shall be decided if the GS should take into account the rotation of the Earth.	T
PC_090	Plot Tools	F	It shall provide tools to plot and export the different metrics.	T
PC_100	DSS-SIM compatibility	F	It shall be able to interact with the DSS-SIM engine.	T
EMULATOR_010	Operation range	F	It shall operate from 400 MHz to 60 GHz.	T
EMULATOR_020	Input power	F	It shall support input power of at least 33 dBm (TBC) look for those devices that allow this power	T

A.2.2. Work packages description

Project: SCE	WP ref.: WP1
Major constituent: Review of the state of the art	Sheet 1 of 5

Brief description: Review of current technologies used to achieve the implementation of the desired functionalities.	Planned start date: 31/01/2022 Planned end date: 01/02/2022	
	Start event: 31/01/2022 End event: 01/02/2022	
Internal task WP1.1: Orbit propagator: Research for already implemented libraries to achieve the propagation of the orbits. Internal task WP1.2: Determination of attenuation, Doppler, and delay: Research for already implemented solutions to achieve the determination of those parameters. Internal task WP1.3: ZeroMQ review: Review of the tool ZeroMQ, what it is and how to use it. Internal task WP1.4: MsgPack review: Review of the tool MsgPck, what it is and how to use it.	Deliverables: -	Dates: -

Project: SCE	WP ref.: WP2	
Major constituent: Design	Sheet 2 of 5	
Brief description: Design the tests to implement.	Planned start date: 14/02/2022 Planned end date: 18/02/2022	
	Start event: 14/02/2022 End event: 18/02/2022	
Internal task WP2.1: Module description: Define the final functionalities that the SCE shall have.	Deliverables: Design report which includes	Dates: 21/03/2022

Internal task WP2.2: List of requirements: Define the requirements that the SCE shall have. Internal task WP2.3: Module high-level architecture: Define the final high-level architecture of the SCE. Internal task WP2.4: Workflow diagram: Define a workflow diagram, where must appear all the functionalities that the must SCE have.		

Project: SCE	WP ref.: WP3	
Major constituent: 1st test	Sheet 3 of 5	
Brief description: Implement a first test to achieve some of the functionalities and requirements designed in WP2.	Planned start date: 14/02/2022 Planned end date: 18/02/2022	
	Start event: 14/02/2022 End event: 18/02/2022	
Internal task WP3.1: Orbit propagation: Achieve the propagation of the orbits Internal task WP3.2: Compute parameters: Apply the review methods to compute attenuation, Doppler and delay. Internal task WP3.3: Connection with local host: Achieve the transmission of the parameters between the PC and the local host.	Deliverables: Report	Dates: -

Project: SCE	WP ref.: WP4

<table>
<tr><td>Major constituent: 2nd test</td><td colspan="2">Sheet 4 of 5</td></tr>
<tr><td rowspan="2">Brief description:
Extend the functionalities of the test performed in WP4.</td><td colspan="2">Planned start date: 14/02/2022
Planned end date: 18/02/2022</td></tr>
<tr><td colspan="2">Start event: 14/02/2022
End event: 18/02/2022</td></tr>
<tr><td>Internal task WP4.1:
Connection PC-SDR: Achieve the connection via sockets between the PC and the SDR. Moreover, the transmission of the parameters must be also performed.

Internal task WP4.2:
Implementation of external tools: Implement the reviewed tools in WP1, in order to extend the functionalities of the network.</td><td>Deliverables:
Report</td><td>Dates:
-</td></tr>
</table>

<table>
<tr><td>Project: SCE</td><td colspan="2">WP ref.: WP5</td></tr>
<tr><td>Major constituent: Final test</td><td colspan="2">Sheet 5 of 5</td></tr>
<tr><td rowspan="2">Brief description:
Perform a final test to achieve the defined requirements.</td><td colspan="2">Planned start date: 14/02/2022
Planned end date: 18/02/2022</td></tr>
<tr><td colspan="2">Start event: 14/02/2022
End event: 18/02/2022</td></tr>
<tr><td>Internal task WP5.1:
Final test: Perform a similar test than the done in WP4, but continuous operation of orbit propagation and parameters calculations is performed.</td><td>Deliverables:
Report</td><td>Dates:
21/06/2022</td></tr>
</table>

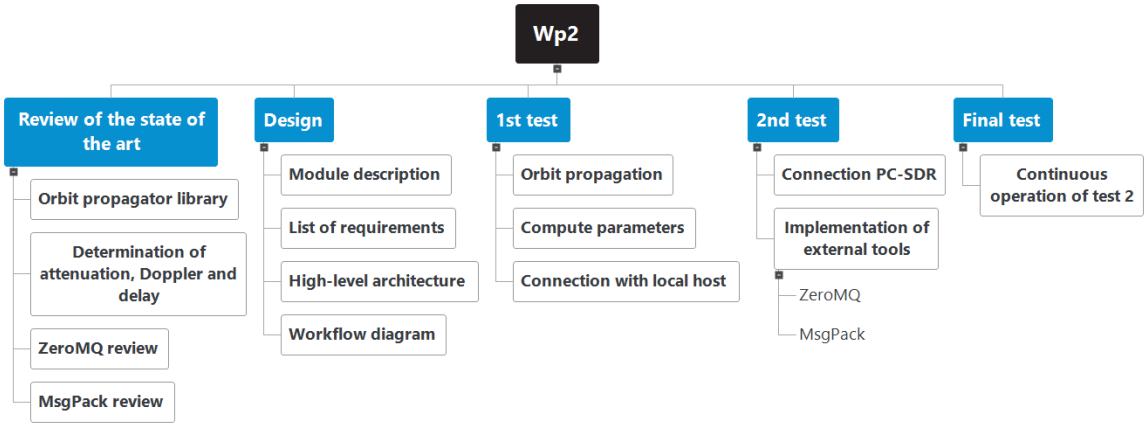


Figure 28: SCE work breakdown structure

A.2.3. Gantt diagrams

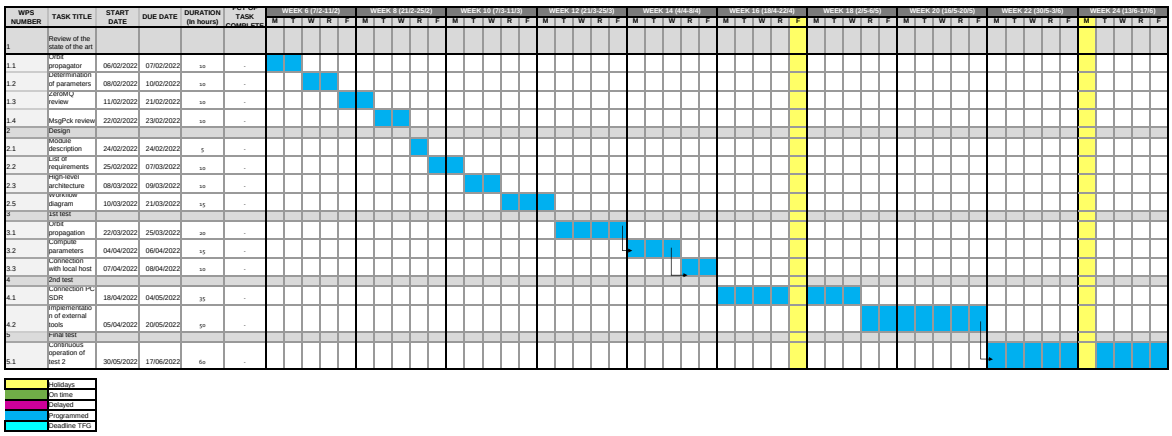


Figure 29: SCE initial Gantt diagram

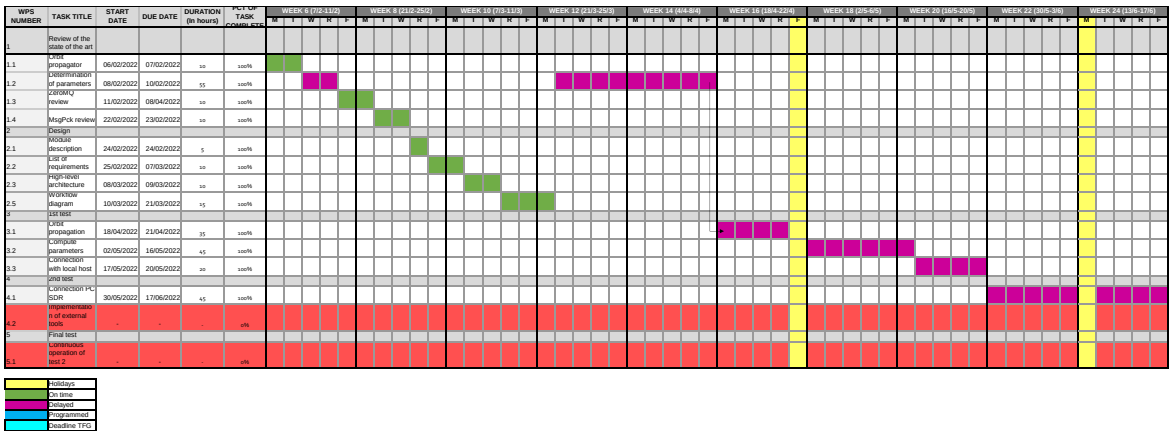


Figure 30: SCE final Gantt diagram

A.2.4. Deviations from initial proposal

The main delay of the project was the calculation of channel effects. After reviewing some information, a first model of them was made, but later it was discarded when testing it. Because of that, more information reviewing had to be done in order to compute the effects accurately. Despite achieving the calculations at the end, the delay has provoked that the continuous transmission of the parameters had not been done. If the calculations of the channel had been done on time, it would have remained time to address the multithreading issues and perform the continuous operation of the emulator.

B. Budget

As there are both projects, the budget will be computed separately. The costs that will be considered are the prototyping cost and the designing cost, which is divided into personal cost and used tools cost.

To compute the tool cost, the amortization of the is needed to be computed. This amortization is computed by following formula:

$$A = \frac{C - R \cdot C}{L} \cdot H \quad \textbf{(16)}$$

Where:

- A = Amortization (€)
- C = Purchase cost (€)
- R = Residual value (%)
- L = Lifspan (h)
- H = Tool usage (h)

However, residual value is null for software tools, as well as for long lifespan hardware tools.

B.1. DSS-SIM

The main constituent of the project is a software, so it has non prototyping cost associated. To compute the cost of the personal it is assumed an average salary of a junior engineer of 24.949 €[20] per year, which leads into a wage of 13,66 €/h, assuming an annual work of 1826 h, which is the maximum annual working hours at Spain[21].

The main software tools used during the project were free software, but the documentation has been done using Microsoft Office tools, which have a price of 69 €/year[22]. Also, a computer has been needed which was provided by i2Cat. The computer itself was a Dell Inspiron 15, whose price is 808,99 €[23] for the version with an i7 processor and 16 GB RAM.

To sum up, the total designing cost has been of 4.287,43 €. The total cost calculation is done in Table 3, Table 4, and Table 5**¡Error! No se encuentra el origen de la referencia.**.

[20] Junior Engineer wage (June, 2022)
[21] Resolución de 13 de enero de 2022
[22] Microsoft 365
[23] Dell Inspiron 15

Table 3: DSS-SIM personal cost

Personal cost		
Activity	**Worked hours**	**Total cost**
WP1	25 h	341,50 €
WP2	30 h	409,80 €
WP3	100 h	1.366,00 €
WP4	85 h	1.161,10 €
WP5	30 h	409,80 €
	Total	3.688,20 €

Table 4: DSS-SIM hardware tools costs

Hardware costs					
Activity	**Worked hours**	**Hardware used**	**Purchase cost**	**Lifespan**	**Hardware cost**
WP1	25 h	Computer	808,99 €	5 years	55,38 €
WP2	30 h	Computer	808,99 €	5 years	66,46 €
WP3	100 h	Computer	808,99 €	5 years	221,52 €
WP4	85 h	Computer	808,99 €	5 years	188,29 €
WP5	30 h	Computer	808,99 €	5 years	66,46 €
				Total	598,10 €

Table 5: DSS-SIM software tools costs

Services costs					
Activity	**Worked hours**	**Service used**	**Purchase cost**	**Lifespan**	**Service cost**
WP1	25 h	-	Free	-	-
WP2	30 h	Draw.io	Free	-	-
WP3	100 h	Visual Studio Code, GitHub	Free	-	-
WP4	85 h	Visual Studio Code, GitHub	Free	-	-
WP5	30 h	Microsoft Office 365	69,00 €	1 years	1,13 €
				Total	1,13 €

B.2. Satellite Contact Emulator

The main constituent of this project is a prototype, so in this case prototyping costs must be computed. As explained in Section 4.2, the final testbed consists of 2xADALM Pluto SDRs[24] and a switch[25]. A computer is also needed to perform the tests, but its use is computed as tools costs.

Unlike DSS-SIM project, no paid software have been used to carry out this project. Regarding personal costs, same salary is considered as in Annex B.1. However, more hardware tools have been used to develop this project, including a signal generator and a spectrum analyzer. Unfortunately, no response has been obtained when asked for

[24] ADALM-PLUTO
[25] Netgear GS108PE-300EUS

quotation, so a price of 4.750[26] € per device has been suppose. The amortization of the products is computed taking into account a total lifespan of 5 years.

The final cost of this project has been 546,83€ for prototyping the last testbed, while the designing cost has been of 5.717,03 €. Th detailed calculations are performed in Table 6, Table 7, and Table 8.

Table 6: SCE prototyping cost

Prototyping cost			
Device	**Price**	**Quantity**	**Total cost**
ADALM Pluto	226,42 €	2	452,84 €
Switch	93,99 €	1	93,99 €
		Total	546,83 €

Table 7: SCE hardware tools cost

Hardware costs					
Activity	**Worked hours**	**Hardware used**	**Purchase cost (combined)**	**Lifespan**	**Hardware cost**
WP1	75 h	Computer	808,99 €	5 years	166,14 €
WP2	40 h	Computer	808,99 €	5 years	88,61 €
WP3	100 h	Computer	808,99 €	5 years	221,52 €
WP4	55 h	Computer, Spectrum Analyzer, Signal generator	808,99 €	5 years	121,84 €
		Spectrum Analyzer	4.750,00 €	5 years	715,36 €
		Signal generator	4.750,00 €	5 years	715,36 €
				Total	2.028,83 €

Table 8: SCE personal cost

Personal cost		
Activity	**Worked hours**	**Total cost**
WP1	75 h	1.024,50 €
WP2	40 h	546,40 €
WP3	100 h	1.366,00 €
WP4	55 h	751,30 €
	Total	3.688,20 €

[26] Spectrum Analyzers Rent - KWIPPED

C. Environmental report

Despite being a final degree thesis focused on software development, some hardware tools have been used during its development, so it has generated some environmental pollution. This pollution is estimated in CO_2 emissions. However, not only hardware devices cause those emissions, but software products also cause them, as the consumed electricity also produces emissions [32].

To compute the CO_2 some products have associated a carbon footprint report, where it is specified the amount of pollution produced during its useful life, including the manufacturing and shipping. This is the case of Dell laptops, which is the one used to develop the project. Despite the exact model of the computer does not provide the carbon footprint, a similar one has been used to estimate the impact.

The computer[27] generates a total amount of 386 kg of CO_2 emissions during its life, being a 91% of them related to manufacturing, transportation, and end of life residues. Knowing this information, the total impact of the computer can be assumed. However, more hardware has been used during the development of the thesis, such as SDRs, switches, signal generators and spectrum analyzers. Unfortunately, none of these products have an associated carbon footprint, so their CO_2 emissions will not be computed. However, it must be kept in mind that they also generate pollution, as well as cloud applications such as GitHub (where the code is stored).

The minimum is 353,44 kg of CO_2 emission. The calculation is done in Table 9. Note that the usage contribution considers the PC used hour, which are less than the total lifespan hours.

Table 9: Computer CO2 emissions

Computer CO2 emissions		
Type	**Contribution**	**Co2 Emissions**
Manufacturing	87,20%	336,59
Usage	9,00%	2,57
Transport	3,30%	12,74
End of life residues	0,40%	1,54
	TOTAL	353,44

[27] Dell Inspiron 16 - Carbon footprint