Development of a Cognitive Assistant for the Preliminary Design of Spacecrafts

by

Sergio Escosa Rodriguez

March 2019

Submitted to the faculty of Escola Superior d'Enginyeria Industrial, Aeroespacial i Audiovisual de Terrassa (ESEIAAT)

of Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

in Partial Fulfillment of the Requirements for the

Bachelor's Degree in Aerospace Engineering and
Bachelor's Degree in Industrial Engineering

Under the guidance of

Daniel Selva Valero, Texas A&M University

Department of Aerospace Engineering
Texas A&M University, College Station, Texas

"There's a tale about the machine against the human, and it creates this popular belief that machines are against us, but when humans and machines work together as allies, not adversaries, we are presented with a great range of new opportunities."

Abstract

The aim of this thesis is to develop a cognitive assistant for the high-level design of a spacecraft bus. The motivation for such a system comes from both the challenges of developing an end-to-end software and the challenges of system architecture in general, specifically satellite design. My work is inspired by the success of Daphne, the first cognitive assistant which helps users in the architecting of space missions, and the possibility of complementing it with a system to assist humans in the design of individual units of a mission. This system has been developed using a layered architecture, the front-end consists of a web interface that serves as a dashboard were the user can edit the current design of a satellite. The parameters that define the design can be introduced into the system in many ways, either by speech recognition, natural language processing or manually editing the design. The back-end layer, serves as the brain of the system, it takes the inputs of the user and evaluates each subsystem of the satellite. It starts by modeling each subsystem with a rule based expert system that then sends suggestions back to the web interface. Ultimately, the user receives these recommendations where it is told which parts of the design could be optimized and which of them are right. The final result of this thesis is the Satellite Design Assistant, an open-source cognitive assistant to support the high-level design of all the subsystems of a satellite by giving feedback and recommendations to the systems engineering team.

Keywords: systems engineering, satellite design, spacecraft, cognitive assistant, aerospace engineering

Resum

L'objectiu d'aquest treball de fi de grau és desenvolupar un assistent cognitiu per al disseny d'alt nivell dels subsistemes d'un satèl·lit. La motivació per crear aquest sistema prové tant del gran repte que ha suposat desenvolupar un *software* de principi a fi, com dels reptes existents en el disseny de sistemes complexos en general, i dels subsistemes d'un satèl·lit en particular. El meu treball ha estat inspirat per l'èxit de *Daphne*, el primer assistent cognitiu que ajuda als seus usuaris en el disseny d'alt nivell de missions espacials d'observació terrestre, a més de la possibilitat de complementar-lo amb un sistema per assistir humans en el disseny de unitats concretes d'un sistema de satèl·lits. El sistema s'ha desenvolupat amb una arquitectura de capes, la primera de les quals és la interfície web, que serveix com a un tauler d'eines on l'usuari pot editar el seu disseny del satèl·lit. Els paràmetres que el defineixen es poden introduir en el sistema de diverses formes, des de reconeixement de veu i processament de llenguatge natural fins a editant-los manualment. La segona capa és el *back-end* o processador, i actua com el cervell del sistema. Primer, agafa els paràmetres introduïts per l'enginyer i avalua cadascun del subsistemes del satèl·lit. Després crea un model de cada subsistema utilitzant un sistema expert basat en regles que envia les suggeriments un altre cop a la pàgina web. Al final, l'usuari rep aquestes recomanacions on se li explica quines parts del disseny es podrien optimitzar i quines ja estan bé. El resultat final d'aquest treball de fi de grau és l'Assistent en el Disseny de Satèl·lits, un assistent cognitiu de codi obert per ajudar als enginyers de sistemes en el disseny d'alt nivell de tots els subsistemes d'un satèl·lit.

Paraules clau: enginyeria de sistemes, disseny de satèl·lits, vehicle espacial, assistent cognitiu, enginyeria aeroespacial

Resumen

El objetivo de este trabajo de fin de grado es el desarrollo de un asistente cognitivo para el diseño de alto nivel de los subsistemas de un satélite. La motivación para crear tal sistema proviene tanto del gran reto que supuso el desarrollo de un *software* de principio a fin, como de los retos existentes en el diseño de sistemas complejos en general, i de los subsistemas de un satélite en particular. Mi trabajo ha estado inspirado por el éxito de *Daphne*, el primer asistente cognitivo que ayuda a sus usuarios en el diseño de alto nivel de misiones espaciales de observación terrestre, además de la posibilidad de complementarlo con un sistema para asistir a humanos en el diseño de unidades concretas de un sistema de satélites. El sistema se ha desarrollado con una arquitectura por capas, la primera de las cuales es una interfaz web, que sirve como tablero de herramientas donde el usuario puede editar su diseño del satélite. Los parámetros que lo definen se pueden introducir en el sistema de varias formas, desde por reconocimiento de voz y procesamiento del lenguaje natural, hasta directamente editándolos manualmente. La segunda capa es el *back-end* o procesador, y actúa como el cerebro del sistema. Primero, coge los parámetros introducidos por el ingeniero i evalúa cada uno de los subsistemas del satélite. Después crea un modelo de cada subsistema utilizando un sistema experto basado en reglas que envía las sugerencias otra vez a la página web. Al final, el usuario recibe estas recomendaciones donde se le explica que partes del diseño se podrían optimizar más y cuáles ya están bien. El resultado final de este trabajo de fin de grado es el Asistente en el Diseño de Satélites, un asistente cognitivo de código abierto para ayudar a los ingenieros de sistemas en el diseño de alto nivel de todos los subsistemas de un satélite.

Palabras clave: ingeniería de sistemas, diseño de satélites,vehículo espacial, asistente cognitivo, ingeniería aeroespacial

Acknowledgements

In the first place, my thanks go to my advisor, Daniel Selva, for helping me develop this project while at the same time helping me adapt to the USA and being a great advisor altogether. His research together with the rest of SEAK Lab have opened my mind to the great field of systems engineering, and his vision of developing intelligent and interactive tools that make the most of the strengths of humans and computers has inspired this project.

In the second place, I would like to thank CFIS, for bringing me the opportunity of doing research in a major American university as well as for all the support received from the staff in the process.

Last but not least, I also want to thank all the different sources of funding which have allowed me to do this project in the USA while only having to worry about my research: the TFG scholarship from CFIS and Fundació Cellex, the MOBINT scholarship from Generalitat de Catalunya, and Texas A&M University for hosting me during this time.

Contents

List of Figures

List of Tables

To my parents and my brother

Chapter 1

Introduction

1.1 Motivation

The engineering design of a spacecraft has enormously progressed over the last decades and now extremely sophisticated techniques are used to assess the optimal solution for a component, subsystem or architecture. This process requires understanding the mission requirements, including payload characteristics and other constraints such as orbit, mission lifetime and performance requirements. Then, the architecture of the mission is chosen, and the subsystems of the different spacecrafts are designed, considering all the requirements and constraints in order to provide the functions necessary for mission success.

It is at this point, where the most important design decisions need to be made regarding the overall architecture of the mission, and it is also at this point where less detailed information is available about what will be the optimal design. Indeed, system architecting remains mostly an art rather than a science, even years after the publication of the foundational work in the field by Rechtin and Maier [25]. The main reason is that it is a task that requires creativity and dealing with deep uncertainty and ambiguity, and these abilities are hard to standardize. However, many parts of the design process have been automated; tools such as simulations and optimization are being used by all engineers in the design process to help them select the best system configuration.

Moreover, traditionally missions have been mainly monolithic spacecrafts with all the instruments placed in one satellite. Nowadays, this trend is changing to building missions which include distributed or fractionated systems, where the mass and the number of instruments of each satellite have been reduced compared to the larger monolithic missions of the early 2000's [38]. Two commonly cited advantages of such distributed architectures are the increase in reliability and robustness achieved

by reducing the number of single points of failure in the system, and improved affordability as individual units become smaller and less costly. On the other hand, distributed systems may lead more complexity in the design of the architecture, and therefore an increase in the cognitive difficulty of the design process.

NASA's technology roadmap for technology area "TA 11: Modeling, Simulation, Information Technology, and Processing", recognizes this problem as it describes a need for improved "Analysis Tools for Mission Design". Specifically, the document states that current tools are thought for monolithic missions and only take into account small parts of the system at a time [30]. Nowadays, large space organizations such as NASA and ESA have design facilities where engineers speed up the mission design process by rapidly assessing the cost, value, feasibility and risk of multiple mission concepts.

Thus, improved tools are needed to architect these complex constellations as there is a need to account or the entire system as opposed to a single satellite during the early design process. These tools help the user to analyze large, high-dimensional design spaces that define the complex space of decisions (tradespace) that can be made when selecting the optimal architecture for a mission. There has been some research in this area [37] [42], including tools currently being developed by NASA, such as TAT-C [29].

To counter the increase in cognitive difficulty, intelligent agents called Cognitive Assistants (CA) have been studied for the last 20 years. Their objective, as well as that of most decision support tools developed, is "augmenting human intellect", as told by D.C Engelbart in one of the first works in human-computer interaction (HCI) back in 1962 [11]. CA aim to facilitate user interaction without overloading it with information, and they make use of modern visualizations to provide the critical information for assisting the user in the decision-making process. Most CA send and receive information to/from the user by means of natural language, either through a voice or text-based interface, and then query an expert system with a knowledge database to provide the necessary information to the user.

This project extends an existing software package for mission design developed at Texas A&M SEAK Lab to address some of its current limitations. The software consists of a Cognitive Assistant (DAPHNE) for aerospace systems architecting [5]. Daphne can take two different roles (Analyst and Critic), draw information from three different sources (historical database, expert knowledge-base, and knowledge extracted from data mining), and communicate with the user through verbal and visual interfaces. As Daphne has a modular structure, we can add new capabilities

and improve the current design of one of its modules. Its different agents are tasked with solving different parts of the problem, such as the design of a specific subsystem, the synthesis of new architectures or the evaluation of the science benefit of each architecture in the tradespace.

Daphne focuses on serving as a tool to help the user in the process of designing the architecture of a mission. Once this choice has been made, and the orbits and payloads of each satellite of the architecture have been assigned, the next step would be the preliminary design of the individual units that are part of the architecture.

As the reader may know, the design of an individual satellite is also considered as a complex system problem. A satellite is the result of the interactions of many subsystems each one performing a determined task to support the payload requirements and influencing the rest of the spacecraft. As mentioned before, large space agencies have design facilities where engineers can speed up the process of designing the different subsystems, they are called Concurrent Design Facilities [4]. There, the engineers from the different subsystems work together in the same room, gathering all the available information and putting in common all their problems and designs that are considered.

Different tools are used in these facilities to assist humans in the process, but this project focuses on developing a tool to help engineers gather all the parameters that define a satellite and give feedback on the design they are working on.

Motivated by the challenges of system architecture in general and architecting an individual satellite in particular, and inspired by the success of Daphne, an opportunity was identified to explore the task of developing a **CA to assist humans in the high-level design of the spacecraft bus**, complementing the skills of Daphne and considering the interrelation with the rest of elements of the mission concept, and assuming that elements like the orbit, launcher or payload have been already decided and are fixed. Thus, this project focuses on two aspects:

- *Improve* the module responsible for designing the different subsystems of the spacecraft bus. These modules use first-order physics-based equations to select the appropriate subsystem components to meet all the mission requirements as well as estimate the mass and size of the whole satellite.
- *Build* a web interface-based assistant where the user can receive feedback of its current design of a satellite.

This report explains in detail how the Satellite Design Assistant was built, both the front-end **web interface** to interact with the user and the back-end **rule based expert**

system which does all the heavy work of designing each subsystem and making the recommendations. It then goes on whit a use case where the software is tested with a real satellite design. Finally, the limitations, future work and conclusions of the project are presented.

1.2 Background

In order to design and develop the mentioned assistant, it was important to review the literature on three main pillars that support the thesis: **the space mission design process, cognitive assistants and expert systems.**

1.2.1 Space Mission Analysis and Design Process

Space mission design usually seeks to find the optimal configuration of a space system which satisfies one or more general objectives and constraints at the lowest possible cost. In other words, it pursues to find the cheapest space system that is able to meet certain mission requirements and offer the best performance at that cost. There are now a number of references available on the mission design process and the definition of mission objectives, but this thesis is based on The Space Mission Analysis and Design Process (SMAD) book [43], which gives a broad view of all the process and a detailed explanation of systems engineering.

Figure 1.1 from SMAD [43] summarizes the space mission analysis and design process that has been widely applied during the last 50 years of space exploration from earth observation missions to interplanetary missions. The analysis and design processes shown in Figure 1.1 are iterative and repeated multiple times for each mission, so that the requirements are improved and refined at every single step, ending in a very well defined space mission concept which can easily end with hundreds of requirements for a small mission.

In *step 1*, the qualitative goals of the mission are defined in a very general way. In *step 2*, it is estimated in a quantitative way how well the objectives of the mission must be achieved, taking into account the needs, technology and budget that the mission has available. These parameters defined at this point must evolve over several iterations during the process. A frequent problem in space mission design is to concrete these requirements too early in the design process, which may after lead to wrong decision making, underperforming designs or too expensive designs for the requirements that had to be fulfilled.

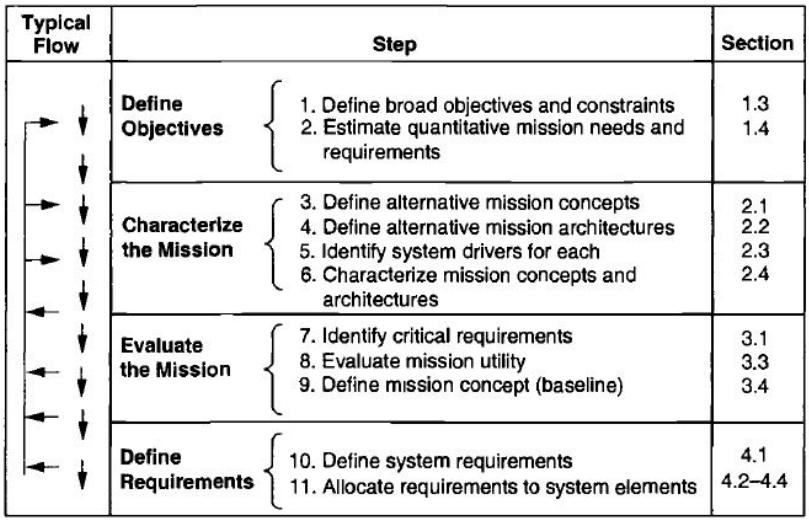

FIGURE 1.1: Mission analysis and Design Process

In *step 3*, mission concepts or concepts of operations are defined. A mission concept is a document or a model that explains how the mission will work in practice and contains information such as which kind of data will be acquired, how it will be sent to the ground, the mission control procedure and a complete timeline of the mission. In *step 4*, different alternate combinations of mission elements are defined (i.e. mission architectures) in order to achieve all the needs detailed in the mission concept. The space mission architecture includes the description of the subject of the mission, the mission operations, the control and communications architecture, the orbit and constellation, the payload, the spacecraft bus, the launch vehicle and the ground segment.

In *step 5*, the cost and performance system drivers are identified (number of satellites, altitude, power, size and weight) for each of the mission concepts and architectures developed in steps 3 and 4. Knowing these driving features and tuning these parameters in particular makes it easier to find the design that perform better at the lowest cost.

Step 6 is one of the most important steps because it characterizes all the system budgets and decides what will be done on board in space and what data will be processed on the ground. In *step 7*, each system defined in the previous stages is evaluated and critical requirements responsible for the cost and complexity of the system are identified. It is important not to confuse critical requirements with system drivers, although they can be strongly related. For instance, coverage (critical requirement) will be strongly related to changes in altitude and, consequently, may become a system driver. *Step 8* consists of mission utility analysis, in which it

is evaluated how well requirements and broad objectives are met as function of cost and/or system design choices. The objective of this step is to provide the decision maker a single chart of potential performance vs cost of the different mission concepts.

Having evaluated alternative designs and done a preliminary assessment of mission utility, in *step 9* one system is selected as a baseline to keep working on. A baseline design is a single consistent definition of the system which meets most or all of the mission objectives. Iteration over iteration, this baseline develops until it becomes the final design.

Finally, in order to know how well we have completed the space mission analysis and design, in *steps 10 and 11* the initial broad mission objectives and constraints are translated into well-defined numerical system requirements and allocate them to the components of the space missions.

1.2.2 Space Mission Life Cycle

Figure 1.2 from SMAD illustrates the life cycle of a space mission, which typically progresses through four phases:

- *Concept exploration* is the initial study phase, which results in a broad definition of the space mission and its components. It is the phase explained in section 1.2.1.
- *Detailed development* is the formal design phase, which results in a detailed definition of the system components and, in larger programs, development of test hardware and software.
- *Production and deployment* consists of the construction of the ground and flight hardware and software and launch of the first full constellation of satellites.
- *Operations and support* are the day-to-day operation of the space system, its maintenance and finally the deorbit or recovery at the end of the mission life.

These phases may be divided and named differently depending on whether the sponsor (the group which provides and controls the budget) is DoD, NASA, ESA or any other organization.

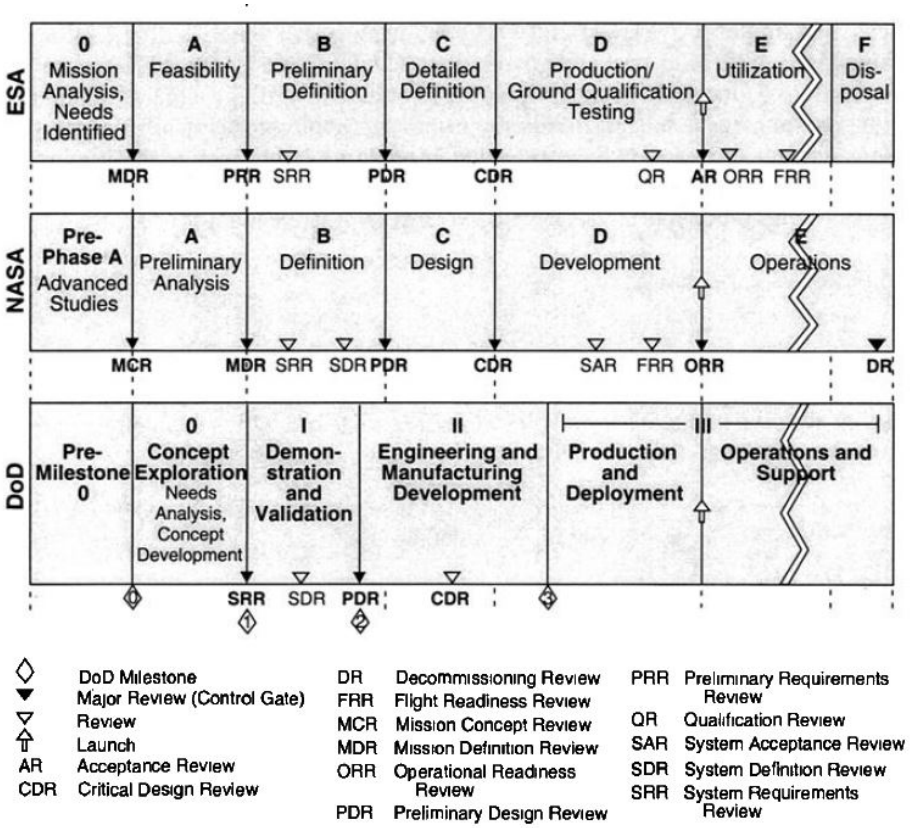

FIGURE 1.2: Mission Cycle at ESA, NASA and DoD

During the Concept Exploration Phase, three basic activities occur. Users and operators develop and coordinate a set of broad needs and performance objectives, as explained before. At the same time, developers generate alternative concepts to meet these needs. In addition, the sponsor performs long-range planning, develops an overall program structure and estimates the budgetary needs. This project and the work done by SEAK Lab at Texas A&M emphasizes in **developing cognitive assistants and tools to help decision makers in this concept exploration phase**, previous to the detailed definition of the mission, with the objective of analyzing and selecting the best design alternatives either on an architecture level (Daphne) or a subsystem level (Satellite Design Assistant).

Figure 1.3 shows two graphs. The first one plots the project expenditures as function of time during the life cycle of a space mission, and the second one plots the percentage of the project cost that has been determined as function of time.

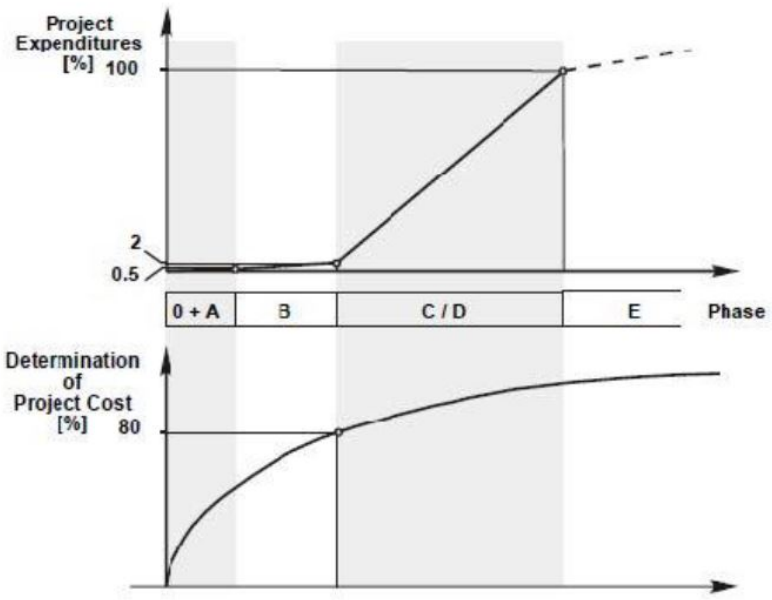

FIGURE 1.3: Mission cost and Percentage of budget determined during each phase.

As Figure 1.3 shows, the most important decisions of the mission design process such as the choice of architecture or the preliminary design of the subsystems must be made when most of the money has not been spent yet, because the detailed design and manufacturing has not started. But more important, nearly 80% of the budget has been already allocated to some phase of the project by the end of the concept exploration phase. That is why, **human assistants are so important in this phase of the mission design process; to help decision-makers have a broad view of the high-dimensional decision space that systems engineering generates.**

1.2.3 Systems Engineering

Systems engineering is an interdisciplinary approach to enable the realization of successful complex systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements and then proceeding with design synthesis and system validation while considering the complete problem. Systems engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that goes from concept to production and operation.

Before going on we should define what is a system. NASA defines it as a set of interrelated components which interact in an organized fashion towards a common

purpose as told in NASA SP-6105 handbook [22]. It can be noted the importance of the interrelationship between the components, and not just designing focusing on a single component. Since all space missions consist of a set of elements or components as shown in Figure 1.4, the arrangement of these elements generates a space mission architecture, which basically is a system of subsystems.

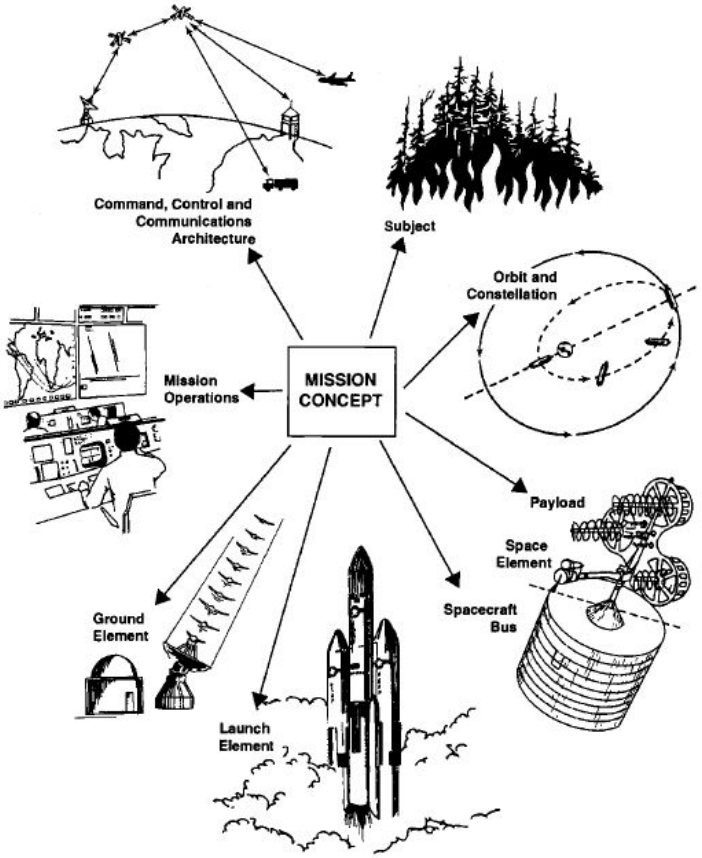

FIGURE 1.4: Elements of space mission architecture.

The *subject* of the mission is the agent which interacts with or is sensed by the space payload: moisture content or atmospheric temperature, for weather missions; types of vegetation or water for Earth observing missions; or a rocket or missile for space defense missions.

The *payload* consists of the hardware and software that sense or interact with the subject. Typically, there is a trade off and combine several sensors and experiments to form the payload. The subsystems of the *spacecraft bus* support the payload by providing orbit attitude maintenance, power, command, telemetry and data handling, structure and temperature control. The payload and the spacecraft bus together are called the spacecraft or space segment.

The *launch system* includes the launch facility, launch vehicle and any upper stage required to place the spacecraft in orbit. The selected launch system constraints the size, shape and mass of the spacecraft.

The *orbit* is the spacecraft's trajectory. Typically, there is a separate initial parking orbit, a transfer orbit and the final mission orbit. There may also be an end-of-life or disposal orbit. The mission orbit influences every element of the mission and provides many options for trades in mission architecture.

The *communications architecture* consists of all the components which satisfy the mission's communication, command and control requirements. It depends strongly on the amount and timing requirements of data to be transferred, as well as the number, location and availability of the space and ground assets used in the mission.

The *ground system* consists of fixed and mobile ground stations around the globe connected by various data links. They allow us to command and track the spacecraft, receive and process telemetry and mission data.

Finally, *mission operations* consist of the people, hardware and software that execute the mission, procedures and data flows.

1.2.3.1 Spacecraft Bus Design

As mentioned before, this project is focused on developing a Cognitive Assistant to help users in the design of the spacecraft bus, considering the interrelation with the rest of elements of the mission concept, and assuming that elements like the orbit, launcher or payload have been already decided and are fixed, and they all determine the mission requirements. For this reason, from now on we will focus on the Design and Sizing of the multiple satellite subsystems: ADCS, communications, power, propulsion, thermal control and structure. All together compose what we know as the spacecraft bus, which is in charge of the following tasks, represented in Figure 1.5:

The *propulsion subsystem* provides thrust for changing the spacecraft's translational velocity or applying torques to change its angular momentum. Most spacecraft need some controlled thrust, so their design includes some form of metered propulsion – a propulsion system that can be turned on and off in small increments. Compressed gasses, such as nitrogen, and liquids, such as monopropellant hydrazine, are common propellants. Significant sizing parameters for the subsystem are the total impulse and the number, orientation and thrust levels of the thrusters.

Subsystem	Principal Functions	Other Names	References
Propulsion	Provides thrust to adjust orbit and attitude, and to manage angular momentum	*Reaction Control System (RCS)*	Sec. 10.4.1, Chap. 17
Attitude Determination & Control System (ADCS)	Provides determination and control of attitude and orbit position, plus pointing of spacecraft and appendages	*Attitude Control System (ACS), Guidance, Navigation, & Control (GN&C) System, Control System*	Secs. 10.4.2, 11.1, 11.7
Communication (Comm)	Communicates with ground & other spacecraft; spacecraft tracking	*Tracking, Telemetry, & Command (TT&C)*	Secs. 10.4.3, 11.2
Command & Data Handling (C&DH)	Processes and distributes commands; processes, stores, and formats data	*Spacecraft Computer System, Spacecraft Processor*	Secs. 10.4.4, 11.3, Chap. 16
Thermal	Maintains equipment within allowed temperature ranges	*Environmental Control System*	Secs. 10.4.5, 11.5
Power	Generates, stores, regulates, and distributes electric power	*Electric Power System (EPS)*	Secs. 10.4.6, 11.4
Structures and Mechanisms	Provides support structure, booster adapter, and moving parts	*Structure Subsystem*	Secs. 10.4.7, 11.6

FIGURE 1.5: Subsystems that compose the spacecraft bus.

The *attitude determination and control subsystem* measures and controls the spacecraft's angular orientation (pointing direction), or, in the case of a guidance, navigation, and control system, both its orientation and linear velocity. The simplest spacecrafts achieve control by passive methods such as spinning or interacting with gravity and magnetic field. More complex systems employ controllers to process the attitude and actuators to change it. The capability of the attitude control subsystem depends on the number of body axes to be controlled, control accuracy and speed of response.

The *communications subsystem* links the spacecraft with the ground or other spacecraft. It is sized by the data rate, allowable error rate, communication path length and RF frequency used.

The *power subsystem* provides electric power for the equipment on the spacecraft and the payload. It consists of a power source, power storage and power conversion and distribution equipment. It is sized based on the power needed to operate the payload and the power duty cycle imposed by the eclipses and peak power consumption. Because solar cells and batteries have limited lives, our design must account for power requirements both at beginning-of-life (BOL) and end-of-life (EOL).

The *thermal control subsystem* controls the spacecraft equipment's temperatures. It does so by the physical arrangement of equipment and using thermal insulation and coatings to balance heat from power dissipation, absorption from the Earth and Sun, and radiation to space. The amount of heat dissipation and temperatures required for the equipment to operate and survive determine the subsystem's size.

The *structural subsystem* carries, supports, and mechanically aligns the spacecraft equipment. It also cages and protects folded components during boost and deploys them in orbit. The main structure is sized by either the strength necessary to carry the spacecraft mass through launch accelerations or stiffness needed to avoid dynamic interaction between the spacecraft and the launch vehicle structures.

1.2.4 Cognitive Assistants

Intelligent tools have been used to support the design of complex systems since the birth of the computing era, like [26] or others as seen in [21]. These tools have taken different forms, which include intelligent Computer Aided Design (CAD) systems [17], knowledge databases [24], design assistants [5] and design critics [20]. Since the focus of SEAK Lab [34], concretely the design of Daphne, the Cognitive Assistant being developed by the Lab at Texas A&M, is centered on the first stages of design, sometimes referred as conceptual design, we are going to focus this part of the background review on this kind of tools.

Intelligent personal assistants are intelligent agents that manage and perform tasks on behalf of humans to reduce routine tasks and the cognitive load of the human users. It is important to note that the main goal of developing these systems is not replacing humans, but to enhance human capabilities both in performing routine tasks and solving complex problems. They usually take the shape of interactive visualizations and decision support tools which allow for the analysis of the different design alternatives, and which have the capacity to handle the thousands of options which can exist for a design problem. Other tools utilize unsupervised machine learning algorithms such as manifold learning, feature selection and clustering to help visualize solutions in a high-dimensional space.

Cognitive Assistants have a long story: starting with NLS from Engelbart back in 1962 [11], there has been a continuous stream of them: RADAR [15], a calendar and email management system that can extract relevant information or CALO [28], a virtual assistant which uses a Belief-Desire-Intention (BDI) model where the user can assign tasks to the agent. More recently, as voice recognition software and natural language processing have advanced to an almost usable level, commercial alternatives have appeared, such as IBM Watson [13,23], Wolfram Alpha [44], Siri [3], Google Assistant [18], Amazon Alexa [2] or Microsoft Cortana [27]. All these systems share the fact that are generalists: they try to answer as many queries as possible from the user using lots of data sources. But generalist CAs are not useful when it comes to a very specialized task, such as aerospace design.

Most CAs in aerospace are thought out to be used by Air Force pilots. Examples of these assistants include CAMA [31], which is an intelligent assistant for ensuring pilot's situational awareness during a flight. A similar type of assistant has been developed to help pilots of multiple UAV systems by telling the pilot when something strange might be happening in one of the missions [9].

Finally, DAPHNE [5] would be closer to a design CA in the sense that all of them are thought out to help the systems engineer come up with a good design. To reduce the cognitive load of systems engineers, one of the main objectives of the tool in this thesis, is to serve as a design critic who offers recommendations of a possible spacecraft bus design. In the future this could complement the Critic Agent of Daphne, by giving feedback on a spacecraft design.

1.2.5 Rule Based Expert Systems

As mentioned in the motivation section 1.1, the back-end of the software, consists of a rule based expert system. This choice was made because at the core of the Daphne CA, sits VASSAR (Value Assessment of System Architectures), a model to assess the scientific value of a system architecture, developed by Daniel Selva [35,36]. VASSAR contains a rule-based expert system to model the knowledge-intensive components of the problem. Thus, due to the experience of my thesis director in the field and of the members of the Lab, we decided to develop the back-end of the Satellite Design Assistant using a rule-based expert system to provide recommendations on the design of a spacecraft bus.

An expert system is "a computer program designed to model the problem-solving ability of a human expert" as told by John Durkin in [10]. In order to do that, an expert system uses large bodies of heuristic – expert knowledge. In a rule-based expert system (RBES), expert knowledge is encapsulated in the form of logical rules. This is in opposition to other kinds of expert systems that primarily use different data structures to store expert knowledge, such as frames in frame-based expert systems (FBES).

In RBES, a logical rule is composed of a set of conditions in its left-hand side (LHS), and a set of actions in its right-hand side (RHS). The actions in the RHS are to be executed if the conditions in the LHS are all satisfied. An example of a logical rule is the following: LHS: = "if the car won't start", RHS: =" then check the electrical engine". An RBES infers information from rules in one of two ways: forward chaining, when logical rules are used from the data to the goal in a deductive

process; backward-chaining, when the rules are applied working backwards from a target goal, as explained by Buchanan & Shortliffe in [7].

RBES consist of three major elements: a fact database, a rule database, and an inference engine. The fact database contains relevant pieces of information about the specific problem at hand called facts. Information in facts is organized according to predetermined data structures similar to C structures and Java Beans, with properties and values. These data structures are called templates in many RBES development tools. Facts can be asserted, modified and retrieved from the database anytime. The rule database contains a set of logical rules that contain the domain knowledge. The LHS of these rules may match one or more facts in the working memory. The RHS of these rules define the actions to be executed for each of these matches, which typically include asserting new facts, modifying the matching facts, performing calculations or showing some information to the user.

The inference engine performs three tasks in an infinite loop: a) pattern matching between the facts and the LHS of the rules in the working memory and creation of activation records (also known as the conflict set); b) while there remain activation records, select the next rule to execute, or fire (conflict resolution); c) execute the selected rules' RHS (deduction). Most current rule engines are based on the Rete algorithm developed by Forgy in 1982 [14]. The Rete algorithm is faster than other algorithms because it "remembers" prior activation records in a network in memory called the Rete network. The Rete network is very efficient in speeding up the search process because most of the time, the network does not change much between iterations. Note that the improvement in computational time comes at the price to increased use of memory.

CLIPS (C-language integration production system) is a public language to write expert systems developed in 1985 at NASA Johnson Space Center as an alternative to the proprietary ART*inference [33]. Ten years later in 1995, Dr Friedman-Hill at Sandia National Labs developed an expert system shell in Java based on CLIPS, especially tailored for RBES, called Jess [16]. As any other RBES, Jess deals with rules and facts, and its inference engine is based on the Rete algorithm. CLIPS/ Jess syntax is very similar to common LISP, one of the earliest programming languages in artificial intelligence developed by Steele [40].

In Jess, the properties in templates are defined using the `deftemplate` command and properties are listed using keywords `slot` (single element attributes) or `multislot` (multiple element attributes). Rules are defined using the `defrule` command. A fact is added into working memory, modified or removed using the `assert`, `modify`

and `retract` commands respectively. Whenever the working memory is modified, the Rete network is recalculated. Matching rules are fired in the order determined by the Rete algorithm once the run command is sent. The general structure of the definitions of a template, a fact, and a rule are provided in the following example. For further information on the Jess language, a good overview can be found in [16].

Jess is the language used to write the rule based expert system of the Satellite Design Assistant developed in this project. Furthermore, Jess is written in Java, which facilitates the integration with other environment, such as Matlab, which will be used to assess some physical models of the subsystems.

1.3 Approach

The objective of this project, as its titles states, is to design and develop the software architecture for a Cognitive Assistant to **assist humans in the high-level design of a spacecraft bus.**

To achieve this objective, the project has been developed inside a group (SEAK Lab, at Texas A&M) which has been working during years in the development of cognitive assistants to help humans in the design of complex aerospace systems. This means that, while most of the design of the software architecture and subsystem models has been developed from scratch, some parts used the work done by the group during these past years.

The rest of this thesis will try to be a general explanation of the whole project and at the same time a description of the parts of the work done by myself. It is organized as follows: Chapter 2 is an overview of the whole system, both the back-end and the front-end. Then, in Chapter 3 two use cases are presented where the assistant could be useful. Finally, Chapter 4 wraps the whole project up while exposing the limitations of the software and future work that could be done to improve it. An appendix is added at the end with the source code.

Chapter 2

Description of the System

This chapter describes the most important part of the work done during the development of the project. It starts with an overview of the architecture of the system and continues explaining more in detail both the front-end and back-end, taking a closer look in the main part of the project, the Subsystem Expert System.

2.1 Architecture Overview

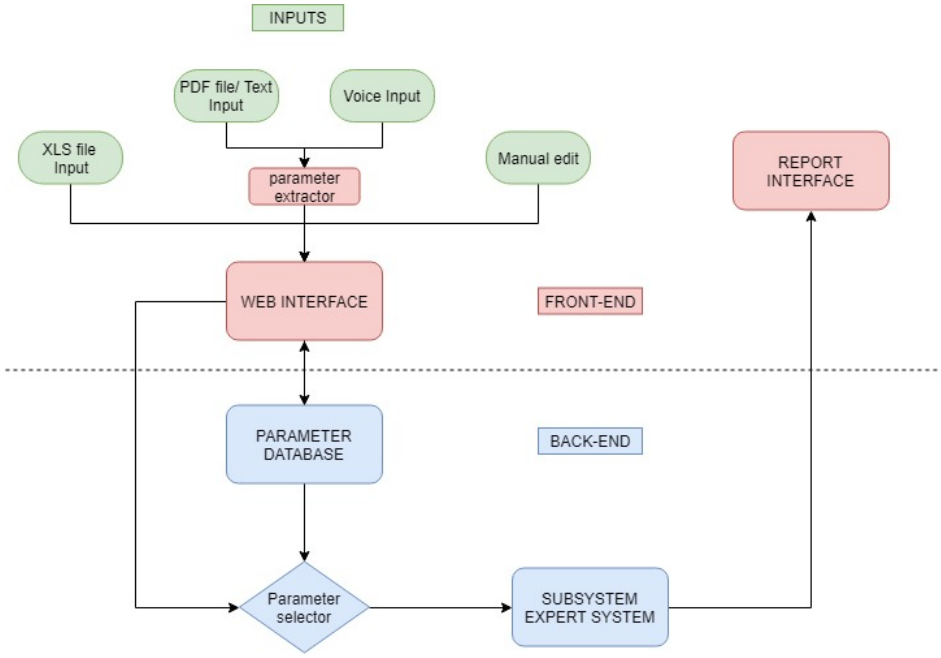

FIGURE 2.1: Flow diagram of the architecture of the Satellite Design Assistant

The Satellite Design Assistant is structured in 2 different layers which interact with each other. At the top we have the different inputs the user can choose to introduce the information of the design into the system. These inputs can be: an Excel file,

containing the parameter and the value; a PDF file or raw text, containing a description of the satellite which is processed by a Natural Language Processing algorithm to extract the relevant information; a Voice description of the design, which is processed to text and sent to the parameter extractor as the text input and finally, the user can also manually edit the parameters directly into the web interface.

Each time the web interface server detects a new parameter, it is added to the database. When all the parameters of a specific subsystem are introduced into the database, the user can evaluate this subsystem. When this request is sent to the server, another module called "parameter selector" extracts the parameters needed to evaluate the specific subsystem requested by the user and are sent to the subsystem expert system.

Finally, the subsystem expert system evaluates the design and provides a series of recommendations on the design described by the user. These recommendations are sent to the web interface and a Report Page is rendered with the output of the system.

This chapter is organized from the bottom up: it starts with the Back-end (section 2.2), explaining more in detail the Parameter Database (section 2.2.1), the Subsystem Rule Based Expert System (section 2.2.2, the QA System (section 2.2.3) and the physical models developed for each subsystem (section 2.2.4). The chapter ends with the Front-end Web Interface (section 2.3), paying more attention to the text input module (section 2.3.1), the file input module (section 2.3.2), the editing dashboard (section 2.3.3) and the reporting page (section 2.3.4). This order helps in understanding the whole system better and also was the chronological order of the development.

2.2 Back-end

In this section, the different parts of the back-end of the system are explained. The back-end does all the heavy work necessary to obtain the recommendations and send them to the front-end and display them to the user. In order to achieve this goal, the system uses different modules that will be explained in the following sections. Firstly, the parameters are stored in a database, then the parameter selector makes a list of the required parameters to evaluate a specific subsystem, this list is used by the Expert System to evaluate the design and write the recommendations.

2.2.1 Parameter Database

The back-end of the webpage is programmed using a Java framework to code the server side of a webpage, called Spring Framework [32]. This framework makes the process of writing the back-end of a webpage easier, by managing the dependencies and has many tools that can be linked between them very easily. One of these tools is the JPA Repository, which is basically a relational database. Its most compelling feature is the ability to create repository implementations automatically, at runtime, from a repository interface, using Java objects that you can use to implement other features of the webpage.

In order to create a parameter database then, a JPA Repository was created containing *Parameter* objects. A *Parameter* object contained the following information:

- *Parameter name*: the name that appears in the webpage.
- *Parameter ID*: a unique number to differentiate the parameters.
- *Slot name*: the name that the parameter has inside the rule based expert system.
- *Value*: the value of the parameter, can either be numeric or not.
- *Unit*: the unit of the parameter in case it has.

To evaluate a specific subsystem, it is necessary to have all the parameters required by the Rule Based Expert System. In order to achieve this, the front-end won't let the user evaluate a subsystem that has missing parameters in the database. Furthermore, there are some subsystems that require parameters from other subsystems, in other words, there are parameters that are used by more than one subsystem. To solve this problem, there is a module that selects the parameters that the Expert System requires to evaluate the subsystem that the user chose.

This is done by creating another table into the relational database. This table consists of the following information:

- *Subsystem Name*: the name of the subsystem the user wants to evaluate.
- *Parameters ID*: list of the parameters necessary to evaluate this subsystem.

When a specific subsystem is requested by the user, the parameter selector looks up the table for the list of IDs to select and creates a list with the Slot name and Value of the selected parameters. This list is then sent to the Subsystem Rule Based Expert System, who will evaluate the design and make the recommendations.

2.2.2 Subsystem Rule Based Expert System

The mentioned list of parameters is received by the Subsystem Rule Based Expert System and it is used in the following way:

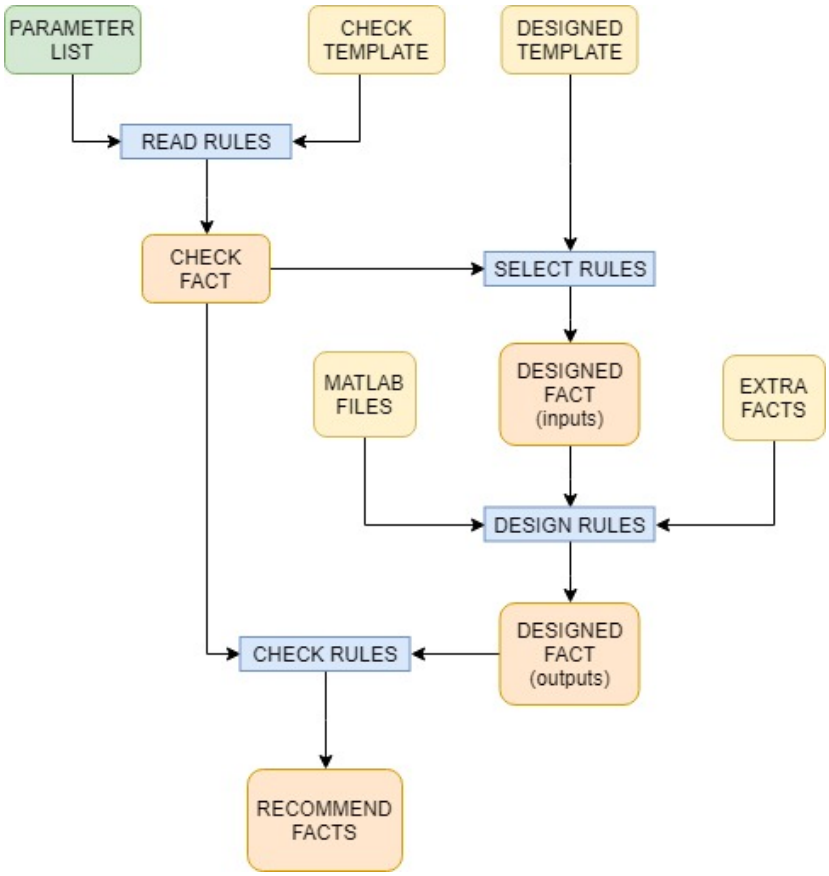

FIGURE 2.2: Flow diagram of the Rule Based Expert System, for a general subsystem

Figure 2.2 shows the flow diagram of the mentioned module, the parameter list is used as an input and the recommendations are the outputs. It is important to mention here that this explanation is the general architecture of all the subsystems, but each subsystem has its design rules and Matlab files. The physical model of each subsystem is explained in section 2.2.4.

As mentioned, this module consists of a rule based expert system that has been written in the JESS language (explained in section 1.2.5). The building blocks of this language are facts and rules and, as consequence, the module consists of these elements.

Each subsystem has a CHECK template and a DESIGNED template to use during the evaluation:

- The CHECK template contains the slots of the different parameters that a specific subsystem requires. When the READ rules fire, they assert a CHECK fact with the parameters available in the list.
- The DESIGNED template contains the same slots as the CHECK template. However, some subsystems will save intermediate results in this fact, reason why the DESIGNED template has more slots than the CHECK template. When the SELECT rules fire, they take the parameters that will be used as inputs from the CHECK fact and copy them into the DESIGNED fact.

When this process finishes, the result is 2 facts: the check fact containing the design of the user, and the designed fact containing only the parameters that will be used as inputs to design the specific subsystem.

To continue with the evaluation, the DESIGN rules will use the input information from the DESIGNED fact plus some other facts depending on the subsystem with other relevant information (i.e. the properties of the different types of solar cells in the power subsystem, or the different types of coatings and insulators for the thermal subsystem). Furthermore, there are some subsystems that also use Matlab files to make some of the calculations. This choice was made because writing mathematical models with Jess language can be sometimes quite hard to do. Moreover, Matlab provides many powerful tools that can be used to improve the model designed by the module (i.e. the Symbolic Toolbox).

Once this rules fire, the DESIGNED fact is completed with the slots that were empty, the output of the model, and can be compared with the same slots but in the CHECK fact that contain the design of the user. Table 2.1 shows an example of these two facts in the power subsystem.

Input slots contain the information that the design rules will use to generate the outputs of each model. The slots that have the output tag are the slots filled with the output of the design rules, but these slots in the CHECK fact are the information provided by the user that will be compared with the same slots in the DESIGNED fact to generate the recommendations. Finally, the auxiliary slots are the ones that are generated by the design rules but are only useful for the model itself to perform future calculations or to provide more detailed information in the recommendations.

The last step happens when the CHECK rules fire. These rules will compare the slots that are tagged with the output type between the CHECK fact and the DESIGNED fact. The rules, take both numbers and compare them with a 10% error, depending on the result of the comparison one recommendation or another will be asserted.

CHECK fact	DESIGNED fact	Slot type
orbit-type orbit-RAAN orbit-altitude payload-power payload-peak-power lifetime satellite-dry-mass solar-cell-type battery-type planet	orbit-type orbit-RAAN orbit-altitude payload-power payload-peak-power lifetime satellite-dry-mass solar-cell-type battery-type planet	inputs
solar-array-area solar-array-mass battery-mass EPS-mass	solar-array-area solar-array-mass battery-mass EPS-mass	outputs
	orbit-period worst-sun-angle fraction-sunlight orbit-semimajor-axis depth-of-discharge satellite-EOL-power satellite-BOL-power battery-capacity	auxiliary

TABLE 2.1: Power subsystem CHECK and DESIGNED facts.

For example, in the power subsystem Electrical Power Subsystem (EPS) mass will be compared between the CHECK fact (*check-mass*) and the DESIGNED (*designed-mass*) fact. If the *check-mass* is higher than the *designed-mass* the system will recommend to the user to reduce the mass of the Power subsystem, since it might be overdimensioned. On the other hand, if the *check-mass* is lower than the *designed-mass*, the system will recommend to the user to make the Power subsystem bigger since it might be incapable of delivering the right amount of energy to the payload and satellite bus. If both numbers are similar (10% error is used), the system will tell the user that the design is correct.

These recommendation facts are used by the back-end of the webpage to render a report page with the information provided by the system.

2.2.3 QA system - Failed approach

This project was started coding the Subsystem Expert System that it is currently being explained. While doing this part it was important to have a module which

would interact with the user. For this reason, a QA system was designed in the back-end to ask the user for missing input parameters and modify the CHECK fact with the answer the user gives. However, when the back-end was integrated with the front-end web interface, it was decided to put this QA system directly into the front-end. The final version of the software, as it will be explained in the front-end section 2.3, tells the user when a parameter is missing in the Parameter Database in the web interface, and does not have to wait until the CHECK facts are created, during the evaluation of a subsystem. This allows the user to receive this feedback without having to wait to evaluate the subsystem.

Although this module is not implemented in the final version of the project, I feel that putting this section in the report is a reminder that research is not always a straight path to success, and for every small step that tries to push forward the state of the art there are a lot of attempts that simply utterly fail. Figure 2.3 describes the mentioned module:

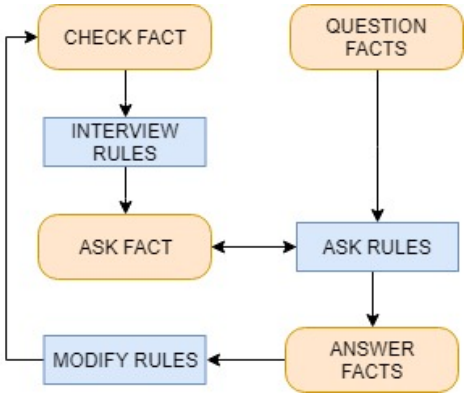

FIGURE 2.3: Flow diagram of the QA system

The module is written in Jess language, which means that it consists of a set of rules and facts. The QUESTION facts contain the questions that the system will ask to the user in case a parameter that has an input tag is missing. The INTERVIEW rules fire when a slot in the CHECK fact is missing and is an input. These rules create an ASK fact with the parameter ID. Then, the ASK rules fire when they match a QUESTION fact and an ASK fact with the same ID, then they ask the user for the required information and assert an ANSWER fact. Finally, the MODIFY rules fire when there is an ANSWER fact with the same ID as the parameter missing in the CHECK fact, then they modify the mentioned fact to save the information of the ANSWER fact, provided by the user.

2.2.4 Spacecraft Bus Physical Model

One of the most important parts of the project that was developed by the author is the work explained in this section. In section 2.2.2, there is a general description of the architecture of the expert system of each subsystem. At the core of these modules there are the *Design Rules* that contain the physical models to obtain the outputs of the model with the inputs provided by the user. The design rules are going to be explained with more detail in this section. Some of this models were taken from the VASSAR [36], but all of them were extended to use more detailed models and to adapt them to the current project.

Each subsystem has a set of the mentioned design rules and Matlab files that take the input parameters from the CHECK fact, obtain the output parameters and save them into the DESIGNED fact. Then, the output values are compared to the values given by the user and the recommendations are generated. Thus, each subsystem will be explained following the same structure:

- Input and Output parameters description
- First-order equations used to model the subsystem
- Recommendations made to the user

It is also important to comment that not all the parameters are explained and listed in the tables. For the sake of writing an understanding report, some parameters have been grouped (i.e Orbit Parameters), and others that don't provide relevant information to the reader have been omitted (i.e the auxiliary parameters), which only are useful to obtain intermediate results. All the parameters grouped as *Orbit Parameters* are the following, and are used to obtain other auxiliary parameters such as *Worst sun angle, Orbit Period, Orbit Semimajor axis* and the *Fraction of Sunlight*:

- *Orbit Type*: type of orbit based on the altitude and other parameters. The options supported by design are: LEO (Low Earth Orbit), MEO (Medium Earth Orbit), HEO (High Elliptical Orbit), SSO (Sun Synchronous Orbit) or GEO (Geosynchronous Equatorial Orbit).
- *Orbit Right Ascension of the Ascending Node (RAAN)*: angle from a reference direction, called the origin of longitude, to the direction of the ascending node, measured in a reference plane. The ascending node is the point where the orbit of the object passes through the plane of reference. In our case the reference direction is Earth's equatorial plane and the and the First Point of Aries is the

origin of longitude. The options supported by design are: DD, AM, Noon, PM, N-A.

- *Orbit Altitude [km]*: distance from the surface of the Earth to the orbit for quasi circular orbits.
- *Orbit Eccentricity*: parameter that determines the amount by which its orbit around another body deviates from a perfect circle. A value of 0 is a circular orbit, values between 0 and 1 form an elliptic orbit, 1 is a parabolic escape orbit, and greater than 1 is a hyperbola.
- *Orbit Inclination [º]*: measures the tilt of an object's orbit around a celestial body. It is expressed as the angle between a reference plane (Earth's equatorial plane) and the orbital plane or axis of direction of the orbiting object.
- *Planet*: planet that the satellite orbits. At the current version of the software, only Earth is available but it can be upgraded to support other planets from the Solar System, as it is explained in Future Work (section 4.2).

2.2.4.1 Power Subsystem

The satellite power subsystem, also known as the Electric Power Subsystem (EPS), is responsible for generating, storing, controlling and distributing electrical power to all the other satellite subsystems and to the payload, in order to allow the whole system to work. Thus, that subsystem consists basically of a power source, generally solar panels to create electricity from sunlight, batteries that store energy during the daylight period and supply energy during the eclipse period and some other components such as regulators and converters. The power subsystem design process must take into account beginning of life (BOL) and end of life (EOL) power requirements since solar cells and batteries have limited lives. The total power needed to operate the satellite and the power duty cycle will determine the area of solar cells and the size and number of batteries and other equipment required.

Inputs and Outputs

Table 2.2 shows the inputs and outputs of the power model, and each of them is explained below the table:

Input Parameters	**Output Parameters**
Orbit Parameters	EPS mass [kg]
Payload Power [W]	Solar array area [m^2]
Payload Peak Power [W]	Solar array mass [kg]
Lifetime [years]	Battery Capacity [Wh]
Satellite Dry Mass [kg]	Battery mass [kg]
Solar Cell Type	
Battery Cell Type	

TABLE 2.2: Power subsystem Table.

- *Payload Power*: The system needs to know the amount of power needed by all the different subsystem in order to compute BOL (Beginning Of Lifetime) and EOL (End Of Lifetime) power requirements, as well as the peak power to consider worst case scenarios.
- *Lifetime*: As solar cells and batteries have limited lives, it is very important to know the duration of the mission in order to take into account the time degradation of these components from BOL to EOL.
- *Satellite Dry Mass*: This parameter is used in the modeling of the different subsystems to calculate an extra percentage of mass proportional to spacecraft total size, to consider the electronics, converters and other extra equipment.
- *Solar Cell and Battery Type*: The system supports different solar cells (GaAs, Si or Multi- Junction) and batteries (NiH2, NiCd) used for the on board power supply and storage. The system will use the user's input to design the power system but will also recommend a better choice of solar cells and batteries in case the design can be optimized.
- *EPS mass*: The total mass of the power subsystem estimated by the model. This parameter is compared with the user's designed and used to make recommendations.
- *Solar array area*: The total area of solar cells required to power the satellite. In order to give more detailed recommendations to the user, this parameter is also used to give feedback to the user.
- *Battery Capacity*: The total battery capacity required to power the satellite during eclipse in orbit. The number of batteries is not considered because the user can choose them depending on the distribution wanted in the spacecraft bus, and would be a decision to be made later in the design.

- *Battery Mass*: From the battery capacity and the battery type, one can obtain the battery mass used to calculate the total EPS mass.

Description of the model

The following part describes the sizing estimation method used in the power subsystem, explaining in detail all the different models and equations used all along the process. All the equations and estimates used are explained in detail in chapter 11.4 of the SMAD book [43].

The first step is to estimate the amount of power required to operate the payload and spacecraft bus, as we have the payload power as an input, we can use the following approximation to estimate the total required power:

$$P_{avg} = \frac{P_{PayloadAvg}}{0.4} \tag{2.1}$$

$$P_{peak} = \frac{P_{PayloadPeak}}{0.4} \tag{2.2}$$

$$P_{day} = 0.8P_{avg} + 0.2P_{peak} = P_{eclipse} \tag{2.3}$$

The power required by the spacecraft during eclipse and during daylight are approximate equal (as suggested by SMAD book) thus, with the eclipse and sunlight time fractions and the efficiencies of the paths, we can compute the amount of power P_{SA} that the spacecraft must produce during daylight in order to power the whole spacecraft for the whole orbit:

$$T_{day} = Period * FracSunlight; \quad T_{eclipse} = Period - T_{day} \tag{2.4}$$

$$P_{SA} = \frac{\frac{P_e T_e}{X_e} + \frac{P_d T_d}{X_d}}{T_d} \tag{2.5}$$

where P_e and P_d are the satellite's power requirements during eclipse and daylight, and T_e and T_d are the lengths of these periods per orbit. The parameters X_e and X_d correspond to the efficiency of the paths from the solar arrays through the batteries to the individual loads and the path directly from the arrays to the loads, respectively. For direct energy transfer or peak-power tracking (two common types of power regulation) those parameters are between 0.6 and 0.65 for the X_e; and between 0.8 and 0.85 for the X_d.

Once we know the power that the spacecraft must produce, we need to determine the amount of power that the solar array will be capable of producing per area unit, in order to find out the solar array minimum surface. The solar cells degrade during lifetime mission, that is way the area will be determined considering the end-of-lifetime (EOL) power.

$$P_{BOL} = P_0 I_d cos(worstSunAngle) \tag{2.6}$$

$$P_{EOL} = P_{BOL}(1 - deg)^{lifetime} \tag{2.7}$$

where P_0 is the power output per unit area with the Sun normal to the surface of the cells, I_d is the inherent degradation due to the design of assembled solar arrays, which are less efficient than single cells because of assembling design inefficiencies, *Worst Sun Angle* corresponds to the worst case scenario of the incidence of the Sun into the panels and $cos(WorstSunAngle)$ is referred to as the cosine loss. The parameter *deg* represents the % of degradation that the solar arrays suffer every year because of thermal cycling in and out of eclipses, micrometeoroid strikes and material outgassing. A standard value for I_d is 0.77 and the values of P_0 and *deg* depend on the type of solar cell that the user chooses.

Using the information obtained from equations 2.5 and 2.7, it is then very simple to estimate the solar array area required to produce the necessary power by a solar array that will have a capability of producing enough power at the mission end of life.

$$A_{sa} = \frac{P_{SA}}{P_{EOL}} \tag{2.8}$$

The mass of the solar array is estimated using the specific power (W/kg) of the solar arrays, which can range from 14 to 47 W/kg at end of life. SMAD uses an approximate value of 25 W/kg to estimate the mass of a solar array:

$$Mass_{sa} = \frac{P_{SA}}{SpecificPower} \tag{2.9}$$

Next step is to size the energy storage part of the spacecraft. To do so, we need to compute the necessary amount of energy stored within a single battery, also known as the capacity (C_r) in Wh:

$$C_r = \frac{P_e T_e}{n(DOD)3600} \tag{2.10}$$

where DOD corresponds to the *Depth of discharge* (estimated by the model based on the type of orbit) and is the percentage of the total battery capacity removed during a discharge period. P_e is the power required to store during eclipse time, T_e

is the eclipse fraction in seconds and n is the battery-to-load transmission efficiency (frequently set to 0.9). Then, using the specific energy density of the battery cell chosen by the user, we can estimate the mass of the battery pack, which the user can divide in as many batteries required by the design:

$$Mass_{bat} = \frac{C_r}{\rho_e} \tag{2.11}$$

Finally, to complete the analysis if the entire power subsystem, the mass of a power control unit, a regulator and converter and the wiring must be computed as well. Again, from the SMAD book [43], we can obtain an equation to estimate these values:

$$Mass_{others} = 0.02P_{SA} + 0.025P_{SA} + 0.02SatelliteDryMass \tag{2.12}$$

$$Mass_{EPS} = Mass_{SA} + Mass_{bat} + Mass_{others} \tag{2.13}$$

Recommendations

Once all the parameters are calculated, the system will compare the following values between the CHECK fact (information given by the user) and the DESIGNED fact (information calculated by the system), and give different recommendations depending on which of them is higher or if they are equal, within an error of 10%:

- **Solar array area:** checks if the area of the solar arrays is correctly calculated.
- **Battery mass:** checks if the mass of the battery (without considering the number of batteries, but the total capacity) is correct.
- **EPS mass:** checks if the total mass of the EPS is correct, considering the wiring, converters and other parts.

2.2.4.2 Communications Subsystem

The satellite communications subsystem is responsible for allowing the spacecraft to communicate uploading and downloading information to and from a ground station or any other space vehicle. Thus, that subsystem consists basically of a transmitter and a receiver (sometimes assembled into a unique component that we call transceiver) plus an antenna. We will often prefer to have an omnidirectional/hemispheric antenna but, depending on the difficulty of closing the link budget and, especially the data rate selected, sometimes it will be necessary to provide the satellite with a more complex directional antenna which, to show another example

of how coupled are the different satellite subsystems, will probably require a more complex and consequently more expensive ADCS subsystem due to higher pointing accuracy necessities.

The information flowing from the ground station to the spacecraft, also known as the uplink or forward link, consists of commands and ranging tones and normally needs a low data rate. On the other hand, the information flowing from spacecraft to the ground station, also known as the downlink or return link, consists of mission status telemetry, ranging tones and mainly data collected by the payload. As the volume of information in the return link is usually much larger than in the forward link, it requires higher data rate and, consequently, this side of the link is the one that normally shapes and sizes the subsystem.

Inputs and Outputs

Table 2.3 shows the inputs and outputs of the communications model, and each of them is explained below the table:

Input Parameters	**Output Parameters**
Orbit Parameters	Comms mass [kg]
Redundancy	Antenna Gain [dB]
Transmitted Power [W]	Antenna Type
Satellite Dry Mass [kg]	Comms peak power [W]
Data per Day [bpd]	
Band	
Modulation	
Number of Ground Stations	
Ground Station information	

TABLE 2.3: Communications subsystem Table.

- *Redundancy*: This parameter is associated to the reliability of the system in case of hypothetical failure. Normally we will set this parameter to 1 for a single-chain architecture or 2 in which every single component is assumed to be duplicated.
- *Transmitted power*: The power, estimated by the user, sent through the antenna to the ground stations.
- *Satellite Dry Mass*: Like in many other subsystems, the communications model needs to know the total satellite dry mass (i.e., bus mass + payload mass) or

at least a good approximation of it. In many spacecraft design models, this parameter is used to calculate the wiring mass of the different subsystems, as it can generally be well approximated by a percentage of the satellite dry mass.

- *Data per day*: The total number of bits of information that the downlink will need to sent per day.
- *Band*: The system calculates the link budget equation using the band provide by the user, but if an optimal solution with another band is available, the system selects the optimal band (UHF, S-Band, XBand or K-Band).
- *Modulation*: The type of modulation (2,4,8,16-PSK) used for the communication between spacecraft and ground station.
- *Number of Ground Stations*: Number of ground stations that the satellite will communicate with. It can go from 1 to 4.
- *Ground Station Information*: The system considers four ground stations often used by NASA missions, namely Wallops, White Sands, McMurdo and Solna. The information used by the system consists of the frequencies of uplink and downlink for the different bands that can be selected. Moreover, the system uses the Orekit Library to calculate the total contact time of the satellite with the ground system.
- *Communications Mass*: Output calculated by the system with total mass of the subsystem.
- *Communications Peak Power*: Maximum power required to operate the communications subsystem, calculated based on the antenna gain and transmitted power required.
- *Antenna Gain*: Gain of the antenna in dB.
- *Antenna Type*: The system considers three different kinds of antenna: Dipole, Patch or Parabolic. The first two types will be generally used in links that require low gains and use UHF or S-band, whereas parabolic antennas will be chosen both for high gain requirements or when we use high frequency (X-band or K-band).

Description of the model

The following part describes the sizing estimation method used in the communications subsystem, explaining in detail all the different models and equations used all

along the process. All the equations and estimates used are explained in detail in chapter 13 of the SMAD book [43].

The first step is calculating the data rate that the inputs chosen by the user require. Using the Orekit Library [8] we can calculate the *Total Access Time* between the spacecraft and the ground station facilities. The user will choose a number between 1 and 4 for the number of ground stations, and the model will assign which ground stations will the spacecraft make contact with. This information is passed to a Java Class which uses a method with the Orekit Library to obtain the *Total Access Time*. This information together with the amount of data that the satellite must send to ground every day (dpd) allows the model to compute the minimum data rate needed:

$$R_b[bps] = \frac{dpd[bitsperday]}{AccessTime[s]} \tag{2.14}$$

The choice of modulation made by the user lets us check if the user has made any mistake in its design. The modulation (M-PSK) essentially depends on the data rate requirement obtained in equation 2.14. The maximum data rate that we can obtain with a certain modulation scheme is driven by the spectral efficiency (γ) and computed by the following equation:

$$Rb_{max} = \gamma BW = \frac{log_2 M}{2} BW \geq R_b \tag{2.15}$$

where BW is the bandwidth chosen by the user. The data rate calculated in 2.14 must always be smaller than Rb_{max}.

Another limitation that the communications subsystem needs to comply is the Link Budget equation. This equation is key to the communication link design since it defines the relationship between wavelength (λ), data rate (R_b), transmitting antenna gain (G_T), transmitted power (P_T), propagation path length (R), receiver noise temperature (T_R) and receiver antenna gain (G_R):

$$\frac{E_b}{N_0} = \frac{P_T G_T G_R \lambda^2}{(4\pi R)^2 k T_R R_b} \geq \left.\frac{E_b}{N_0}\right|_{\text{min}} \tag{2.16}$$

where k is Boltzmann constant.

$\left.\frac{E_b}{N_0}\right|_{\text{min}}$ is the minimum energy per bit to noise power spectral density ratio, also known as the "SNR per bit" is an important parameter in digital communications or data transmission and it is defined by the modulation scheme chosen and the bit error rate (BER) performance needed. Thus, the link budget equation lets us

approximate the *SNR per bit*, which must always be bigger than the minimum SNR per bit obtained from the modulation and BER. For a typical value of BER = 10^{-5}, and modulations of 2,4,8 and 16 - PSK respectively, we obtain a minimum SNR per bit of:

$$\left.\frac{E_b}{N_0}\right|_{\text{min}} = [10.6\ 10.6\ 14\ 18.3] \text{ for a BER} = 10^{-5}$$

At this moment we have to limits for our communications subsystem, one imposed to the data rate by the modulation and bandwidth; and another imposed to the minimum SNR per bit by the modulation and bit error rate. However, there is a third limit imposed by the physics of the communication link; the shannon limit. The Shannon limit or Shannon capacity of a communications channel is the theoretical maximum information transfer rate of the channel, for a particular noise level, and is defined as follows:

$$\frac{E_b}{N_0} \geq \frac{2^{\eta}-1}{\eta} \text{ where } \eta = \frac{R_b}{BW} \tag{2.17}$$

With all the parameters imposed by the user and these equations, the model will evaluate the user's design and check if all of the three limits are complied. If any of the limits is not true, then the system will tell the user which one is failing and make a recommendation to change some of the input parameters.

If all of the three limits are met, then the system will proceed to evaluate the antenna gain and type chosen by the user and try to find an optimal solution for the link budget equation that defines the communication. The first step is to evaluate if the gain of the antenna can be minimized, because an antenna with a lower gain will have a simpler design and, as consequence, less cost and weight. Therefore, we will calculate and try to minimize the margin that the SNR per bit has until it meets any of the lower limits, either the shannon limit or the minimum SNR per bit:

$$\text{Margin} = \frac{E_b}{N_0} - \left.\frac{E_b}{N_0}\right|_{\text{min}} \tag{2.18}$$

In order to minimize this equation we will try to find if there exists any value for the antenna gain that makes the margin equal to 4 dB (we leave a small margin as a conservative measure). If there is any value that would minimize the margin, this gain will be chosen by the model for the transceiver, and the system will find an antenna type which meets this new requirement. The system will also recommend the user to stay with the current margin and change other inputs such as the data rate, the bandwidth or the modulation.

There is the possibility that even with an antenna gain of 0 dB, the margin would still

be positive. In this case, the system will recommend other options to the user such as using a higher bandwidth or modulation, a higher data rate. The system will also show the user a design with an antenna that has a gain of 0 dB and the respective margin.

The following simplified code describes the model that this system uses to choose the most suitable antenna type:

```
if Gain < 3 && Band == UHF
    AntennaType='Dipole';
elseif (Gain<9) && Band == {UHF, Sband or Xband }
    AntennaType='Patch';
else
    AntennaType='Parabolic';
end
```

When the model has chosen an antenna gain and an antenna type it will go to the next step which is sizing the antenna. Each antenna type (Dipole, Patch or parabolic) has a sizing model which defines its dimensions and weight. These models can be found in the SMAD book [43]. For example, the diameter and mass of a parabolic antenna can be estimated with the following equations:

$$Diameter = D = \sqrt{\frac{G_T \lambda^2}{\pi^2 eff}} \tag{2.19}$$

$$Mass = \rho\pi \cdot \frac{D^2}{4} \cdot \frac{(1+4\beta^2)^{1.5}-1}{6\beta^2}(1+\alpha) \tag{2.20}$$

where eff corresponds to the antenna efficiency, ρ to the material density, β to the parabolic aspect ratio and α to the support to dish mass ratio.

Once the system has chosen a more optimal antenna gain and antenna type, the next step consists in estimating the power and mass of the communications system. From [19], the mass of these components is derived as a function of the transmitting power (P_T) following a simplified linear extrapolation of existing components. The transmitter incorporates a TWTA (traveling wave tube amplifier) and the modeling allows to have either only a transmitter or both a transmitter and amplifier. The equations area the following:

$$Mass = Mass_{trans} + Mass_{amp} + Mass_{wiring} = \tag{2.21}$$
$$= (0.008\frac{P_T}{eff_{trans}} + 0.5) + (0.005\frac{P_T}{eff_{amp}} + 2) + 0.01 \cdot SatelliteDryMass$$

where $eff_{trans} = 0.1$ and $eff_{amp} = 0.7$ according to [1].

Then, the total mass of the communications subsystem is:

$$Mass_{comms} = Mass_{antenna} + Mass_{electronics} \tag{2.22}$$

Another important output parameter from the communications system is the peak power required by the subsystem to allow every single component to work. This value is computed using the following equations, that can be found in the SMAD book.

$$Power_{peak} = P_T \cdot (1 + \frac{1}{eff}) \tag{2.23}$$

$$\text{where } eff = max[0.45, 0.45 - 0.015 \cdot (P_T - 5)]$$

Recommendations

Once all the parameters are calculated, the system will compare the following values between the CHECK fact (information given by the user) and the DESIGNED fact (information calculated by the system), and give different recommendations depending on which of them is higher or if they are equal, within an error of 10%:

- **Link budget equation:** check if the calculated SNR per bit is higher than the shannon limit and the minimum SNR per bit.
- **Data rate :**check if the data rate required by the user is less than the maximum data rate.
- **Margin Analysis :**mentioned, the margin of the link budget equation can be minimized, and this minimum can be 0 or be greater than zero. In this recommendation, the system explains to the user which is its situation showing the calculate margins and antenna gains, and possible actions to minimize the margin without changing the gain, such as, changing the band or having a higher data rate.
- **Antenna analysis :**the system shows to the user its current choice of antenna and if it finds a possible design with an antenna of lower gain, the system suggests to the user this more optimal design, and the corresponding TTC mass and power.

2.2.4.3 Thermal Control Subsystem

The thermal control subsystem or TCS is responsible of maintaining all spacecraft and payload components and subsystems within their required temperature limits for each mission phase. Temperature limits include a cold temperature which the component must not go below and a hot temperature that it must not exceed. Frequently, there are two limits defined in each end: the *operational limits* that the component must remain within while operating and the *survival limits* that the component must remain within at all times, even when not powered. Exceeding survival temperature limits can result in permanent equipment damage as opposed to out of tolerance performance when operational limits are exceeded.

Thermal control techniques are broadly divided into two categories. *Passive thermal control* makes use of materials, coatings or surface finished (such as blankets or second surface mirrors) to maintain temperature limits. *Active thermal control*, which is generally more complex and expensive, maintains the temperature by some active means, such as heaters or thermo-electric coolers. In general, low-cost thermal control systems are designed to keep spacecraft at the cool end of allowable temperature ranges. Cooler components generally last longer, and this allows for system power growth.

The design process for the thermal control system is very complex and can consider physical models with very high-degree of precision like computational simulations of the heat flow in the satellite. In this project, a simpler model is considered. First, the heat inputs are characterized throughout the entire life of the mission. The most important external great source will nearly always be the Sun, which continuously provides 1367 W/m^2 at the mean distance of the Earth from the Sun. This input disappears whenever the spacecraft enters a period of eclipse during its orbit. However, the Earth or other nearby central body serves as a moderating thermal influence by radiating heat in the infrared.

As it will be explained afterwards, our model determined the radiator, insulator and heater requirements considering a hot case, where the satellite is between the Sun and the Earth during its period, and a cold case where the Earth is between the Sun and the satellite (eclipse).

Inputs and Outputs

Table 2.4 shows the inputs and outputs of the thermal control model, and each of them is explained below the table:

Input Parameters	Output Parameters
Orbit Parameters	TCS mass [kg]
Satellite Dry Mass [kg]	TCS power [W]
Satellite Dimensions [m]	TCS type
Maximum Operational Temperature [ºC]	
Minimum Operational Temperature [ºC]	
Maximum internal heat [W]	
Minimum internal heat [W]	
Radiator surface material	
Insulator surface material	

TABLE 2.4: Thermal Control subsystem Table.

- *Satellite Dimensions*: Dimensions of the satellite in the following reference frame: direction X (direction of the velocity vector of the satellite), direction Y (normal to plane XZ) and direction Z (direction from the satellite's center of gravity to Earth's center).
- *Satellite Dry Mass*: Like in many other subsystems, the thermal subsystem model needs to know the dry mass of the satellite to compute the mass of additional equipment like the heater pipes and others as a percentage of the dry mass of the satellite.
- *Temperature Range*: The operational and survival ranges of the satellite's components. In order to simplify the model, the system sizes the subsystem with the operational limits of the most restrictive component, which means that all the other components will fall inside the range.
- *Internal Heat*: The heat released by the payload and other subsystems into the spacecraft during normal operation. It is used as an input heat to the model.
- *Radiator and Insulator surface materials*: The name of the material used as a coating or surface finish for the radiator and insulator. This information contains the emissivity and absorptivity of these materials.
- *Number of Ground Stations*: Number of ground stations that the satellite will communicate with. It can go from 1 to 4.

- *TCS type*: The system decides if the spacecraft requires a *passive* control system or an *active* control system.
- *TCS mass*: The total mass of the thermal subsystem calculated by the model.
- *TCS power*: The total power that the heater requires calculated by the model.

Description of the model

The following part describes the sizing estimation method used in the thermal control subsystem, explaining in detail all the different models and equations used all along the process. All the equations and estimates used are explained in detail in chapter 11.5 of the SMAD book [43].

The model starts assuming that the thermal control subsystem is passive, meaning that there isn't any heater, and then makes the corresponding calculations to see if a heater is required. In that case, the system will change to active and estimate the required power for the heater.

As mentioned, the first step consists of considering a hot case and a cold case and determine if the heat equilibrium in the satellite is inside the limits. For the **hot case**, the following assumptions are considered:

$$T = T_{max}$$
$$\text{Earth's albedo} = 38\%$$
$$Q_{internal} = Q_{max}$$
$$Qin_{radiator} = Q_{sun} + Q_{internal}$$
$$Qin_{insulator} = Q_{sun} + Q_{albedo} + Q_{IR}$$

where it is assumed that the radiator is facing the Sun and half of the insulator's area is facing the Sun and the other half is facing the Earth. T_{max} is the maximum operational temperature, $Q_{internal}$ is the internal heat released by the payload and spacecraft bus, Q_{sun} is the heat absorbed from the Sun (Very Near Infrared Radiation), Q_{albedo} is the heat absorbed from the Earth's albedo, and the Q_{IR} is the infrared radiation absorbed from the Earth.

In this step, we aim to find the radiator and insulator area required to achieve equilibrium at the required temperature. It is assumed that the 90% of the satellite's surface is covered either with insulator or radiator material. Thus, we can write a

system of 2 equations to determine both areas:

$$\begin{cases} dq = |Q_{out} - Q_{in}| = 0 \\ A_{insulator} = 0.9 \cdot TotalArea - A_{radiator} \end{cases} \tag{2.24}$$

where Q_{out} and Q_{in} depend only on the area variables and other parameters, and can be estimated using the following equations, where Q_{out} consists of radiation heat fluxes and Q_{in} consists of the absorbed heat fluxes described above.

$$Q_{out} = Qrad_{radiator} + Qrad_{insulator}$$
$$Q_{in} = Qin_{radiator} + Qin_{insulator}$$

Now we know both the *Insulator Area* and *Radiator Area* that are required to be safe in the hot case. Then, we evaluate the design in the cold case of the orbit, where we aim to obtain the *Temperature* at which equilibrium is accomplished with the estimated areas. The assumptions for the cold case are the following:

$$Q_{internal} = Q_{min}$$
$$\textit{Earth's albedo} = 23\%$$
$$Qin_{radiator} = Q_{internal}$$
$$Qin_{insulator} = Q_{sun} + Q_{albedo} + Q_{IR}$$

where it is assumed that the radiator is pointing to deep space and half of the insulator's area is facing Earth and the other half is facing deep space.

In the cold case, we aim to find the equilibrium temperature that is reached. Thus, we want to solve the following equation:

$$dq = |Q_{out} - Q_{in}| = 0 \tag{2.25}$$

where Q_{out} and Q_{in} depend only on the equilibrium temperature and other parameters, and can be estimated using the following equations, where Q_{out} consists of radiation heat fluxes and Q_{in} consists of the absorbed heat fluxes described above.

$$Q_{out} = Qrad_{radiator} + Qrad_{insulator}$$
$$Q_{in} = Qin_{radiator} + Qin_{insulator}$$

Once we have estimated the temperature of the cold case, we can compare it with the minimum operational temperature of the spacecraft and decide if a heater is required. If the cold temperature is higher than the minimum temperature, a passive thermal control subsystem would be possible. On the other hand, if the cold temperature is lower, a heater will be required for those situations and thus, the TCS will be active. In this case, we need to estimate the required power of the heater.

The active case, makes the same assumptions as the cold case, but the internal heat is the unknown variable, and the temperature is an input:

$$T = T_{min}$$
$$Earth's\ albedo = 23\%$$
$$Q_{internal} = P_{heater}$$
$$Qin_{radiator} = Q_{sun} + Q_{internal}$$
$$Qin_{insulator} = Q_{sun} + Q_{albedo} + Q_{IR}$$

and the equation we solve is the following:

$$dq = |Q_{out} - Q_{in}| = 0 \tag{2.26}$$

where Q_{out} and Q_{in} depend only on unknown variable $Q_{internal} = P_{heater}$. As the reader might notice, if the variable $Q_{internal}$ is calculated using this method, it won't take into account the internal heat sources that come from the payload and other subsystems. This is done that way because the heater is sized in a worst case scenario were the payload and spacecraft bus are not powered and the only heat source is the heater, in order to size it for an emergency scenario.

Once we have sized the TCS, we can estimate the mass of the subsystem using the approximations explained in the SMAD book:

$$Power_{TCS} = P_{heater} \tag{2.27}$$
$$Mass_{TCS} = Mass_{insulator} + Mass_{electronics} + Mass_{radiator} \tag{2.28}$$
$$Mass_{insulator} = 0.73 \cdot A_{insulator}$$
$$Mass_{electronics} = 0.2$$
$$Mass_{radiator} = 3.3 \cdot A_{radiator}$$

Recommendations

Once all the parameters are calculated, the system will compare the following values between the CHECK fact (information given by the user) and the DESIGNED fact (information calculated by the system), and give different recommendations depending on which of them is higher or if they are equal, within an error of 10%:

- **Heater power:** checks if the system requires a heater, and the power of it.
- **TCS type:** whether the choice between active and passive is correctly done.
- **TCS mass:** checks the total mass of the TCS, considering other equipment such as pipes and pumps.

2.2.4.4 Propulsion Subsystem

The function of the propulsion subsystem is to provide the necessary thrust to all the increments of velocity, changes in orbit, or orbit maintenance operations and attitude control during the lifetime of the mission. Specifically, the model used in this project for the propulsion subsystem computes all the deltaVs required during the life of the mission and size the amount of propellant and mass of the propulsion system based on this calculations. Thus, the model will evaluate different scenarios where an increment of velocity is required, and as a consequence, the current subsystem would be used to achieve the required thrust:

- *Injection to orbit*: The system computes the delta-V required for injection for GEO or MEO assuming a transfer orbit with a perigee of 150km and an apogee at the desired orbit.
- *Overcome drag*: The system computes the delta-V required to overcome drag while in orbit, to maintain the altitude of the orbit.
- *Attitude control*: compute the amount of delta-V required by the ADCS subsystem in order to maintain a specific pointing direction.
- *End of mission*: compute the amount of delta-V requried to put the satellite into a graveyard orbit at the end of mission or to deorbit the satellite to burn it in the atmosphere.

Inputs and Outputs

Table 2.5 shows the inputs and outputs of the propulsion subsystem model, and each of them is explained below the table:

Input Parameters	Output Parameters
Orbit Parameters	DeltaV [m/s]
ADCS control type	Propulsion Mass [kg]
Propellant for orbit injection	Satellite Wet Mass [kg]
Propellant for ADCS thrusters	Satellite Dry Mass [kg]
Payload power [W]	Propellant Mass [kg]
Satellite Dimensions [m]	
Deorbiting Strategy	

TABLE 2.5: Propulsion subsystem Table.

- *ADCS control type*: In order to compute the increment in delta-V due to attitude control, the model uses the ADCS control type. If the control type is gravity gradient, the attitude control subsystem won't need any increment of delta-V, and if it is a three axis control system, delta-V will be set to a positive value.
- *Propellant*: The type of propellant (solid, hydrazine or LH2) used by the thrusters of the injection to orbit propeller and by the ADCS thrusters. This choice determines the specific impulse of each thruster.
- *Paylaod Power*: The propulsion model needs to know the payload power or at least a good approximation of it. In many spacecraft design models, this parameter is used to estimate the wet mass of the satellite.
- *Satellite Dimensions*: Dimensions of the satellite in the following reference frame: direction X (direction of the velocity vector of the satellite), direction Y (normal to plane XZ) and direction Z (direction from the satellite's center of gravity to Earth's center). This parameter is used to evaluate the drag that the satellite sees.
- *Deorbiting strategy*: Type of deorbiting chosen for the end of the mission. It can be either putting the satellite into a graveyard orbit or deorbit the satellite and burn it in the atmoshpere.
- *DeltaV*: Total increment of velocity required during the whole lifetime of the mission.

- *Propulsion mass*: Total mass of the thrusters and other equipement of the propulsion subsystem.
- *Satellite Wet Mass*: Output obtained by the model by adding the satellite dry mass and the propellant mass. Is the mass of the satellite at launch.
- *Satellite Dry Mass*: In the case of this subsystem, the satellite dry mass is an output, and is estimated with DeWeck's algorithm (explained later in this section). This output is very useful to check if the satellite dry mass (input) provided by the user in other subsystems is a good estimate or not.
- *Propellant Mass*: Mass of propellant required to achieve the calculated total delta-V.

Description of the model

The following part describes the sizing estimation method used in the propulsion subsystem, explaining in detail all the different models and equations used all along the process. All the equations and estimates used are explained in detail in chapter 17 of the SMAD book [43].

The first step of the method consists of estimating de total *DeltaV* that the spacecraft will need along its lifetime. To do this, different cases were thrust is required have been considered, and then they are all added together:

The *Injection to Orbit DeltaV*, estimates the delta-V requried for injection into a GEO or MEO orbit, assuming a transfer orbit with a perifee of 150km and an apogee at the desired orbit, as suggested in De Weck's paper [39]. For LEO and SSO, no injection is required. The equations used are the following:

$$\Delta V_{inj} = |V_{orbit2} - V_{orbit1}| \tag{2.29}$$

where V_{orbit} is the orbtial velocity of an orbit with semimajor axis a and distance of the satellite to the focus r and is defined by the following equation:

$$V = \sqrt{\mu_{Earth} \cdot \left(\frac{2}{r} - \frac{1}{a}\right)}$$

The *Drag DeltaV*, computes the delta-V required to overcome drag. The model comes from De Weck's paper [39], and estimates the parameter H_p, to later obtain

the approximated ΔV:

$$H_p = \frac{\text{SemiMajor Axis} \cdot (1 - \text{Eccentricity}) - R_{Earth}}{1000} [\frac{m/s}{year}] \tag{2.30}$$

which is a model that first determines a parameter (H_p) based on the orbit geometry. Then, De Weck proposes the following rule-based model to adapt it to the satellite drag based dV

```
if Hp < 500
        dV = 12;
elseif Hp < 600
        dV =5;
elseif Hp < 1000
        dV = 2;
else
        dV = 0;
end
```

Last, the dV estimated (which is per year), is multiplied by the lifetime of the mission to obtain the total drag based dV.

$$\Delta V_{drag} = dV_{deWeck} \cdot Lifetime$$

The *ADCS DeltaV* computes the delta-V required for attitude control. It is estimated using the following rule of DeWeck:

```
if ADCS = 'Three Axis'
        dV = 20;
elseif ADCS = 'Gravity Gradient' or 'Spinner'
        dV =0;
end
```

Lastly, the dV estimated (which is per year), is multiplied by the lifetime of the mission to obtain the total ADCS based dV.

$$\Delta V_{adcs} = dV_{deWeck} \cdot Lifetime$$

The last case where thrust could be used is for the end of mission disposal. The satellite could be put into an orbit graveyard or could burn into the atmoshpere.

Both methods require an increment of velocity, so first the model decides which method is required based on the orbit type:

```
if orbit = 'GEO' or 'MEO'
        deorbit = 'graveyard';
elseif orbit = 'LEO' or 'SSO'
        deorbit = 'drag-based';
end
```

Then, the deltaVs are estimated. For the *drag based* method, which calculates the ΔV to go from the actual orbit to an orbit with the perigee at Earth's surface. For the *graveyard* method, the method computes the ΔV required for deorbiting the satellite into an orbit with a preige raised by a certain amount. Basicly, a similar procedure as in equation 2.29 considering the desired orbits.

Lastly, the total ΔV is calculated adding all the partial results:

$$\Delta V_{total} = \Delta V_{adcs} + \Delta V_{deorbit} + \Delta V_{drag} + \Delta V_{injection} \tag{2.31}$$

The next step is sizing the propulsion system for the required Δ_V. Again, we will use the model proposed by De Weck, which uses the following algorithm to compute the propellant mass and the satellite wet mass, then substracting the propellant from the wet mass, we can obtain the dry mass of the satellite, which will be useful for estimating the mass of the thrusters as a percentage of the drymass:

```
Wet_mass = 4.6*payload_power^0.73 + 140; %first estimate of
   the wet mass to initialize the algorithm
error = 1000;
while error > 0.1
        mass_prop = rocket_equation_mi_to_mp(dV_inj, Isp_inj,
            Wet_mass) + rocket_equation_mi_to_mp(dV_rest,
           Isp_rest, Wet_mass);
        mw1 = 38*(0.14*Ppower + mass_prop)^0.51;
        error = abs(wet_mass - mw1);
        wet_mass = mw1;
end
```

which starts but calculating an initial estimate of the wet mass of the satellite based on the payload power and then keeps iterating over the algorithm until the solution converges and the error is low. To calculate the propellant masses, the algorithm uses

the rocket equation, which calculates the requried propellant mass for a propellant with Isp, the required dV and the initial mass (wet mass). It can be noticed that it is done with two different dV and two different Isp; that is because usually the injection thrusters are different from the thrusters used in the ADCS, or the orbit maintenance and also usually have different propellants.

When the algorithm converges, we have a good estiamte of the propellant mass of the two types of propellants, the satellite's wet mass and the satellite's dry mass. Then, an estimate found in the literature (SMAD), can be used to estimate the mass of the thrusters:

$$Mass_{propulsion} = Mass_{thruster} + Mass_{propellant} \tag{2.32}$$
$$Mass_{thruster} = 0.04 \cdot \text{DryMass}$$

Recommendations

Once all the parameters are calculated, the system will compare the following values between the CHECK fact (information given by the user) and the DESIGNED fact (information calculated by the system), and give different recommendations depending on which of them is higher or if they are equal, within an error of 10%:

- **Deorbiting strategy:** checks if the choice between graveyrad or drag-based deorbiting is correct.
- **Dry mass:** checks if the satelliet dry mass (which is an output in this subsystem) is coherent with the required wet mass and propellant required by the propulsion design.
- **Propellant mass:** checks if the user's value for mass of propellant is coherent with the calculations of the model. The wet mass is not required to be check because it can be simply done by adding the dry mass and the propellant mass.
- **Propulsion mass:** checks if the mass of the thrusters is correct.

2.2.4.5 ADCS Subsystem

The attitude determination and control subsystem (ADCS) stabilizes the vehicle and orients it in desired directions during the mission despite the external distubrance torques acting on it. This requires that the vehicle determine its attitude, using sensors, and control it, using actuators. The ADCS often is tightly coupled to other

subsystems on board, especially the propulsion subsystem. For this reason, and because it's very difficultl to size accurately the ADCS in the preliminary phase of design, the model used in this thesis is very simple and qualitative, it's probably the most qualitative of all the subsystems. The model uses rules of thumb to design the type of ADCS and the actuators used.

Any body in space is subject small but persistent disturbance torques (i.e. 10^{-4} Nm) from a variety of sources. These torques are categorized as cyclic, varying in a sinusoidal manner during an orbit, or secular, accumulating with time, and not averaging out over an orbit. These torques would quickly reorient the vehicle unless resisted in some way. An ADCS system can resist these torques passively, by exploiting inherent inertia or magnetic properties to make the disturbances stabilizing and their effects tolerable (i.e. gravity gradient), or actively, by sensing the resulting motion and applying corrective torques (i.e. reaction wheels).

Conservation of vehicle angular momentum plays an important role in this subsystem. Angular momentum is conserved unless an external torque is applied, meaning that external disturbances must be resisted by external control toruqes (i.e. thrusters or magnetic torquers) or the resulting momentum buildup must be stored internally (i.e. by reaction wheels).

Inputs and Outputs

Table 2.6 shows the inputs and outputs of the ADCS model, and each of them is explained below the table:

Input Parameters	**Output Parameters**
Orbit Parameters	ADCS mass [kg]
ADCS accuracy [º]	ADCS control type
Satellite Dry Mass [kg]	Slew control method
Moments of inertia [kg m^2]	Number of Reaction Wheels
Slew angle [º/s]	
Satellite Dimensions [m]	

TABLE 2.6: ADCS subsystem Table.

- *ADCS control type*: In order to compute the increment in delta-V due to attitude control, the model uses the ADCS control type. If the control type is gravity gradient, the attitude control subsystem won't need any increment of delta-V, and if it is a three axis control system, delta-V will be set to a positive value.

- *ADCS accuracy*: Degrees of accuracy required to the ADCS. A high accuracy (lower numeric value in degrees) means that the actuator needs to be more precise when applying the torque because the satellite has to be able to move more precisely.
- *Satellite Dry Mass*: Like in many other subsystems, the thermal subsystem model needs to know the dry mass of the satellite to compute the mass of additional equipement like the heater pipes and others as a percentage of the dry mass of the satellite.
- *Satellite Dimensions*: Dimensions of the satellite in the following reference frame: direction X (direction of the velocity vector of the satellite), direction Y (normal to plane XZ) and direction Z (direction from the satellite's center of gravity to Earth's center). This parameter is used to evaluate the drag that the satellite sees.
- *Moments of Inertia*: They are determined within the same directions as the dimensions.
- *Slew angle*: Rate of change in the pointing direction required to the ADCS. A higher number means that the satellite needs to move faster from one pointing direction to another. Thus, the ADCS will be sized differently.
- *ADCS mass*: Total mass of the sensors, actuators and other equipement of the ADCS.
- *ADCS control type*: Type of control system used in the ADCS. The model chooses the most suitable system based on the accuracy needed. The supported systems are: gravity gradient, spinner and three-axis control system.
- *Number of Reaction Wheels*: In the case that the model decides that the most suitable system is the three-axis control, which is the system in most modern satellites, it will consider the number of reaction wheels, which need to be equal to the number of body axis directions that want to be controlled plus some number of backup wheel (usually there is one extra wheel in the most restrictive body axis direction).

Description of the model

The following part describes the sizing estimation method used in the attitude determination and control subsystem, explaining in detail all the different models

and equations used all along the process. All the equations and estimates used are explained in detail in chapter 11.1 of the SMAD book [43].

The model that sizes the ADCS is probably the most qualitative of all the subsystems. Maing decisions and accurate approximations of the parameters that define the ADCS is a difficult task, and usually this subsystem is sized when more information is fixed during the deisgn process. Despite that, our model tries to give as much feedback as possible to the user regarding the choice of control system, the set of sensors to be used and approximates the mass of a *three axis* ADCS, which is probably the most common.

Firstly, the system chooses the slew control method to change the attitude of the satellite based on the slew angle or angular velocity required by the user. The rules used are the following:

```
if slew_angle = 0
        slew_control = 'none';
elseif slew_angle < 0.5
        slew_control = 'thrusters';
else
        slew_control = 'two-force-thrusters';
end
```

where the difference between the two types are that two force thrusters are able to regulate the amount of thrust to make changes in attitude faster.

Then, the type of ADCS is decided based on the accuracy required, using the following set of rules:

```
if accuracy < 1
        adcs_type = 'three-axis';
elseif accuracy < 5
        adcs_type = 'spinner';
else
        adcs_type = 'gravity-gradient';
end
```

where *Gravity gradient* is a passive technique which uses the inertial properties of a vehicle to keep it pointed toward the Earth. This relies on the fact than an elongated object in a gravity field tends to align its longitudinal axis through the Earth's center. This technique is used on simple spacecrafts which require not a

lot of pointing accuracy. *Spinner* is a passive control technique in which the entire spacecraft rotates so that its angular momentum vector remains approximately fixed in inertial space. This systems employ the gyroscopic stability effect to passively resist distrubance torques about two-axes. *Three-axis* refers to techniques where the spacecraft is stabilized in 3 axes. They maneuver and can be stable and accurate, depending on their sensors and actuators, but they are also more expensive and more complex. The control torques about the 3 axis systems come from combinations of momentum wheels, reacion wheels, thrusters or magnetic torquers.

The next step consists on choosing a set of sensors that can determine the attitude of the satellite based on the accuracy required by the designer. To accomplish this task, our model sets the best possible sensor for every possible case using the following rule:

```
if accuracy < 0.01
        sensor = 'star-sensor';
elseif accuracy <0.25
        sensor = 'horzion';
else
        sensor = 'sun';
end
```

This information will be used to recommend a list of possible actions to the user, one of them will be the best choice of sensors for the required accuracy.

Finally, the model sizes the ADCS for a three axis standard design of 4 reaction wheels (one for redundancy). In case the ADCS doesn't require a three-axis tecnhique and *spinner* or *gravity gradient* are chosen by the model, they will be set an approximate fixed mass of 1 kg as they are passive techniques that consist of simple designs that can be accomplished with the shape of the satellite or a torque produced by the launch system respectively. Therefore, to size a three-axis stabilized satellite we need to compute all the disturbance torques that affect the spacecraft and size the reaction wheels to provide the required counter-torque:

The disturbance torque induced by the gravity gradient of the spacecraft can be computed as follows;

$$T_g = \frac{3}{2} \cdot \mu_E arth \cdot \frac{1}{a^3} \cdot (I_z - I_y) \cdot sin(2\theta) \tag{2.33}$$

where $\mu_E arth$ is the gravitational parameter of the Earth, a is the semimajor axis of the orbit, I_z and I_y are the moments of inertia in the respective directions and θ is

the maximum deviation of the Z-axis from local vertical in radians (or the maximum slew-angle that will perform the satellite).

The next equation computes the aerodynamical disturbance torque, induced by the small atmospheric density that is still present in orbit:

$$T_{aero} = \frac{1}{2} \cdot \rho(h) A V^2 C_d \cdot (C_p - C_g) \tag{2.34}$$

where $\rho(h)$ is the density of the atmosphere as function of the altitude, A is the surface area that sees the velocity, C_d is the drag coefficient, V the orbital velocity, C_p the center of aerodynamic pressure and C_g the center of gravity.

The disturbance torque induced by the solar pressure is computed using the following equation:

$$T_{sp} = \frac{F_s}{c} \cdot A_s \cdot (1 + q) cos(i) \cdot (C_{sp} - C_g) \tag{2.35}$$

where F_s is the solar constant, 1367 W/m^2, c is the speed of light, $3x10^8 m/s$, A_s is the surface area, C_{ps} is the location of the center of solar pressure, C_g is the center of gravity, q is the reflectance factor (ranging from 0 to 1, we use 0.6), and i is the angle of incidence of the Sun.

The last disturbance torque consider is the one induced by the magnetic field of the Earth, and can be computed as follows:

$$T_m = D \cdot \frac{2M}{R^3} \tag{2.36}$$

where D is the residual dipole of the vehicle in $A \cdot m^2$, M is the magnetic moment of the Earth, $7.96x10^{15} Tesla \cdot m^3$ and R is the radius from the Earth's center to the spacecraft in m.

Once all the disturbances are computed, we use the maximum disturbance as a worst case scenario to size the reaction wheels. Firstly, we need to calculate the momentum storage capacity that a reaction wheels needs to have to compensate for a permanent sinusoidal distrubance torque that accumulates over 1/4 of the period:

$$h = \frac{1}{\sqrt{2}} \cdot T_{max} \cdot \frac{1}{4} \cdot Period \tag{2.37}$$

Then, we can estimate the mass of a reaction wheel from its momentum storage capacity:

$$M_{RW} = \frac{3}{2} \cdot h^{0.6} \tag{2.38}$$

We can also estimate the mass of the sensors from its accuracy requirement. It is based on data from SMAD chapter 10 page 327:

$$Mass_{sensor} = 10 \cdot Acc^{-0.316} \tag{2.39}$$

Then, we can compute the total mass of the ADCS, considering a standard design with 4 reaction wheels and 3 sensors:

$$Mass_{adcs} = 4 \cdot Mass_{RW} + 3 \cdot Mass_{sensor} + 0.01 \cdot \text{DryMass} \tag{2.40}$$

To compute the power of the system, we can use the estimations from the SMAD book, which suggest that sensor power can be neglected and the power of a reaction wheel can be approximated by the following equation:

$$Power_{adcs} = 3 \cdot Power_{RW} = 3 \cdot (200 \cdot T) \tag{2.41}$$

where we use 3 reaction wheels here because the fourth woud only be used in case one of the others fails, and T is the maximum torque that the reaction wheel can apply and can be approximated as $T = \omega h$. Where h is the system's angular momentum, and ω is the angular velocity required by the user (slew angle).

Recommendations

Once all the parameters are calculated, the system will compare the following values between the CHECK fact (information given by the user) and the DESIGNED fact (information calculated by the system), and give different recommendations depending on which of them is higher or if they are equal, within an error of 10%:

- **Slew control system:** Checks if the choice for the control of the slew angle is correct, based on the slew rate.
- **ADCS type:** Based on the accuracy required, check if the system chosen by the user would meet this requirement.
- **Number of reaction wheels:** checks the number of reaction wheels, consdiering that there is always an extra one for redundancy.
- **ADCS mass:** checks the total mass of the ADCS, considering other equipment such as sensors and wiring.

- **Report:** Since this subsystem has a more qualitative design, the system explains to the user some of the critical design points of the deisgn such as which are the best sensors to use and the problem of the saturation of reaction wheels.

2.2.4.6 Structure Subsystem

The structures subsystem mechanically supports all other spacecraft subsystems, attaches the spacecraft to the launch vehicle, and provides for ordnance-activated separation. The design must satisfy all strength and stiffness requirements of the spacecraft and of its interface to the booster. The primary structure carries the spacecraft's major loads and the secondary structure supports wire bundles, propellant lines, nonstructural doors, and other components that may be folded during launch.

The launch vehicle is the most obvious source of structural requirements, dictating the spacecraft's weight, geometry, rigidity and strength. The launch vehicle, the selected orbit, and upper stage determine the spacecraft's allowable weight. The launch-vehicle structure will determine the necessary rigidity and strength of the structure through the natural frequencies that respond to forces from both internal (engine oscillations) and external (aerodynamic effects) sources. The launch vehicle contractor will list known natural frequencies for each launch vehicle and describes associated axial, bending and lateral modes.

When designing a structure, we have to consider all the optional materials, types of structures and methods of construction. By far the most commonly ised metal for spacecraft structure is aluminium alloy. Alluminium is relatively lightweight, strong, readily available, easy to machine and the raw material is cheap.

The structural model of this thesis considers different types of Alluminium alloys and makes a design decision between a monocoque structure and a stringers structure. Monocoque structures, which are panels and shells without stiffening members, are used only if applied and reacted loads are spread out rather than concentrades. In case that is necessary to increase the buckling strength, the model will opt for a semimonocoque structure, which uses stringers to stiffen the primary structure. Stringers are longitudinal members that accept more concentrated loads and spread them into the skin.

Inputs and Outputs

Table 2.7 shows the inputs and outputs of the structure model, and each of them is explained below the table:

Input Parameters	Output Parameters
Orbit Parameters	Structural Mass [kg]
Satellite Wet Mass [kg]	Structure type
Satellite Length [m]	
Satellite Maximum Diameter [m]	
Axial Load Factor	
Lateral Load Factor	
Bending Load Factor	
Axial Natural Frequency [Hz]	
Lateral Natural Frequency [Hz]	
Structure Material	
Solar array area [m^2]	
Solar array mass [kg]	
Number of Solar Arrays	

TABLE 2.7: Structure subsystem Table.

- *Satellite Wet Mass*: In order to compute the effects of the launcher into the structure, the model has to know the mass of the satellite at launch.
- *Load Factors*: A multiple of weight on Earth, representing the force of inertia that resists acceleration. The load factor applies in the direction opposite that of the acceleration.
- *Factors of safety*: These are not an input parameter because their values are hardcoded into the model. The usual values used are 1.1 for the *Yield Safety Factor* and 1.25 for the *Ultimate Safety Factor*. They are the factors apploed to the limit load to obtain a design with lower chances of failure.
- *Natural Frequencies*: The frequency at which a system tends to oscillate in the absence of any driving or damping force. The *axial* and *lateral* natural frequencies are the ones that respond to the internal and external forces of the launch-vehicle system. This parameter is always provided by choice of launcher.

- *Satellite Dimensions*: Dimensions of the satellite in the following reference frame: the length corresponds to the vertical longitude of the satellite along the direction of launch, and the diameter to the maximum transversal distance inside the fairing, when the satellite has all its components folded.
- *Structure material*: The material from which the structure is made of. This parameter holds the informatoin of the material relative to the poisson coefficient, the young modulus, the density, the ultimate strength and the yield strength. The available materials are: Aluminium (Al7075), Steel (17-4PH), Magnesium (AZ31B) and Titanium (Ti-6Al-4V).
- *Solar array Information*: In order to size the secondary structure that holds the solar arrays, the model needs informaiton about the mass and area of the solar arrays, as well as the amount of panels that are distributed around the structure.
- *Structure mass*: Total mass of the primary structure and the secondary structure, dimensioned to hold the solar arrays during launch.
- *Structure Type*: The model chooses the optimal design between a *monocoque* structure and a *semimonocoque* structure with stringers which accepts a higher buckling load in expense of more weight.

Description of the model

The following part describes the sizing estimation method used in the structure subsystem, explaining in detail all the different models and equations used all along the process. All the equations and estimates used are explained in detail in chapter 11.6 of the SMAD book [43].

As mentioned, the model will choose between a monocoque design or a semimonocoque design, based on the one that, after sizing it for standing the launch loads, wights less. Therefore, the model starts assuming a monocoque design and sizing it. In order to do this, the shape of the main structure is approximated to a cylinder with uniform thickness. Then, we will size the thickness of it for different restrictions and use the most restrictive design (the maximum thickness).

Axial Rigidity: for axial rigidity, we can use the following equation to calculate the cross-sectional area of the structure, that meets the natural frequency requirements:

$$f_{nat} = 0.25\sqrt{\frac{AE}{mL}} \tag{2.42}$$

where E is the Young Modulus, m is the mass of the satellite and L its longitude in the vertical direction of the launcher. Then, from this first equation we can obtain the area, A, and calculate the thickness using the cylindrical shape: $A = 2\pi R t_1$, where t_1 is our first thickness.

Then, for *lateral rigidity*, we use the following equation to estimate the thickness that meets the lateral natural frequency requirements:

$$f_{nat} = 0.56\sqrt{\frac{EI}{mL^3}} \tag{2.43}$$

where we obtain I, the cylinder area moment of inertia, and the required thickness can be calculated using the definition of the area moment of inertia for a cylinder: $I = \pi R^3 t_2$.

The next step is calculating the applied and equivalent axial loads. By mulitplying the spacecraft weight by the load factor, we can derive the limit or maximum expected loads:

$$\begin{aligned}
\text{Axial Load} &= m \cdot g \cdot LF_{axial} \\
\text{Lateral Load} &= m \cdot g \cdot LF_{lateral} \\
\text{Bending Load} &= m \cdot g \cdot \tfrac{L}{2} \cdot LF_{bending}
\end{aligned}$$

Then, we can estimate the equivaent axial load using:

$$P_{eq} = P_{axial} + \frac{2M}{R} \tag{2.44}$$

where M is the Bending Load and R is the radius of the cylinder. Finally, we can multiply this limit load by the ultimate factor of safety to obtain the ultimate load, and by the yield fator of safety, to obtain the yield strength:

$$\begin{aligned}
\text{Ultimate Load} &= P_{eq} \cdot 1.25 \\
\text{Yield Load} &= P_{eq} \cdot 1.1
\end{aligned}$$

Now, we can size for tensile and yield strength, using the material's allowable tensile (F_{tu}) and yield (F_{ty}) strengths, and using the equation for axial stress, $\sigma = P/A$:

$$t_3 = \frac{\text{Ultimate Load}}{2\pi R F_{tu}} \tag{2.45}$$

$$t_4 = \frac{\text{Yield Load}}{2\pi R F_{ty}} \tag{2.46}$$

The last step, is sizing the cylinder for stability (compressive strength) or buckling. We first take the most restrictive thickness calculated until now, by using the maximum of $[t_1, t_2, t_3, t_4]$. Then, we can estimate the critical buckling load using the following eqautions:

$$\phi = \frac{1}{16}\sqrt{\frac{R}{t}}$$
$$\gamma = 1 - 0.9 \cdot (1 - e^{-\phi})$$
$$\sigma_{cr} = 0.6\gamma \cdot \frac{Et}{R} \tag{2.47}$$
$$P_{cr} = A \cdot \sigma_{cr} \tag{2.48}$$

The next step, is checking if the ultimate load applied to the structure is greater than the critical buckling load. To do this, structural integrity is often shown in terms of the *margin of safety* (MS), defined as

$$MS = \frac{\text{Allowable Load or Stress}}{\text{Applied Load or Stress}} - 1 \tag{2.49}$$

and must be greater than or equal to zero. If this is not the case, we will have to make the structure thicker, to meet the buckling restriction. The model does this using the following piece of code which iterates the model until finding a thickness which makes de *margin of safety* (MS) equal to zero:

```
while MS < 0
    tnew = t + 5e-5; %we make the thickness bigger
    phi = 1/16*((D/2)/t_new)^0.5;
    gamma = 1 - 0.901*(1 - exp(-phi));
    sigma_cr = 0.6*gamma*E*t_new/(D/2); % critical buckling
       stress
    A = 2*pi*(D/2)*t_new; %area new
    Pcr = A*sigma_cr; %Critical Buckling Load
    MS = Pcr/UltimateLoad -1; % if MS < 0, increase thickness
    t = t_new;
end
```

Finally, when the code converges, we can estimate the total mass of the main structure:

$$Mass_{monocoque} = \rho \cdot 2\pi R t L; \tag{2.50}$$

The second part consists of sizing for a semimonocoque structure. Suppose we stiffen the cylinder with 12 longitudinal members, caled stringers, and 11 circumferential rings, or frames. With the following code, we can obtain the distance of each stringer to the netural axis of the structure:

```
s = 12; %number of stringers
bays = 10; %number of separations of the cylinder (bays)
f = bays + 1; %number of frames
theta = 360/s; %angle of separation between stringers
d = []; %distance from neutral axis to stringer
for i = 1:s
    d(i) = (D/2)*abs(sin(pi/180*theta*(i-1)));
end
```

Then, we start assigning a minimum thickness to the skin of 0.127 cm which is a result from the SMAD book that is adequate to withstand the acoustic noise. First, we must choose whether to design the skin to help sustain load or whether to allow it to buckle, forcing the stiffeners to take on more of the burden. In our model, we will design the skin not to buckle, as is usually done when performing preliminary sizing analysis.

Again, we will first size for stiffness of the structure. Thus, if our chosen thickness of 0.127 cm is smaller than the required thickness, t_1, calculated before for axial stiffness, we will change the thickness for t_1. Then we size for the bending rigidity by calculating the required area moment of inertia, I_{skin}, of the cylinder's skin:

$$I_{skin} = \pi R^3 t$$

Thus, we can calculate the contribution to the total inerta, I, calculated in the first part (remains constant), of the 12 stringers:

$$I_{stringers} = I - I_{skin}$$

and we can calculate the I of the 12 stringers in the cylinder using the parallel axis theorem, $I_{xx} = \sum(I_{cm} + Ad^2)$. We can ignore the I_{cm} because it will be very small compared to the other term. Therefore, the I of the stringer system is a function of stringer cross-sectional area, A, and d, the distance from the cylinder's neutral axis, calculated before.

The next step is considering panel stability, which is done considering the following equation to determine the compressive buckling stress for the skin panel:

$$\sigma_{cr} = \frac{k\pi^2 E}{12(1-\nu^2)} \cdot \left(\frac{t}{b}\right)^2 \tag{2.51}$$

where $k = 55$, from Fig. 11-35 of the SMAD book, ν is Poisson's ratio and b is the width of the panels. The buckling load, $P_{cr} = \sigma_{cr} A$, lets us calculate the margin of safety MS, for this design:

$$MS = \frac{\text{Allowable Load or Stress}}{\text{Applied Load or Stress}} - 1$$

Again, we must iterate the model like we did with the monocoque design, until we find a thickness that makes the MS greater or equal than zero. Finally, we estimate the mass of the stringers design as follows:

$$Mass_{stringers} = (A_{skin} + A_{stringers}) L\rho \cdot 1.25 \tag{2.52}$$

where a 25% of extra mass is added to count for the ring frames fasteners. The last step is to choose one of the designs, this is simply done by choosing the one that weights less:

$$Mass_{mian} = min[Mass_{stringers}, Mass_{monocoque}] \tag{2.53}$$

Lastly, our model also makes an approximation of the mass of an important secondary structure which is present in mostly all satellite's, the mass of the solar array structure. Its function is to hold the solar arrays during the whole mission, but specially during launch, where the highest loads are applied to the spacecraft.

The sizing process is the same as it was done with the monocoque and, therefore, won't be explained again. We first size for axial and lateral rigidity to meet the natural frequencies requirements. Then we calculate the equivalent axial loads and size for the tnesile and yield strength. Finally, the most restrictive thickness is chosen (the highest), and the mass is estimated as follows:

$$Mass_{SAstruc} = numberSA\rho \cdot Width \cdot Length \cdot t \tag{2.54}$$

The only differences is that the natural frequencies are doubled from the ones of the main structure, in order to make this secondary structure uncoupled from the main one, and dynamically stable. Also, the code to calculate the dimensions of the solar array panel is the following:

```
SA_unit_area = SA_area/number_SA;
SA_mass = SA_mass_total/number_SA; %mass of 1 solar array
l = 10; %slenderness ratio of individual solar array (length/
   wide)
width = (SA_unit_area/l)^0.5;
length = SA_unit_area/width;
```

where the solar array mass, solar array area and number of solar arrays are inputs of the user. Obviously, the mass of the whole structure will be the sum of the main and secondary structure:

$$Mass_{struc} = Mass_{SAstruc} + Mass_{main} \tag{2.55}$$

Recommendations

Once all the parameters are calculated, the system will compare the following values between the CHECK fact (information given by the user) and the DESIGNED fact (information calculated by the system), and give different recommendations depending on which of them is higher or if they are equal, within an error of 10%:

- **Structure mass:** takes into account main structure and solar array structure.
- **Structure type:** check whether the choice between monocoque or semimonocoque is correctly done.

2.2.4.7 Launcher Subsystem

The launch process can severly constrain spacecraft design. Primary restrictions are the launch vehicle's lift capability and the environment to which it subjects the satellite during ascent. A launch system consists of a basic launch vehicle incorporating one or more stages and an infrastructure for ground support. It increments the velocity and altitude of the spacecraft to place it in orbit, while creaing a severe ascent environment, and protects the spacecraft during the process. Ultimately, it places the payload into the desired orbit, or into a transfer orbit, with a functional spacecraft attitude.

The model of this subsystem consists of two parts. The first one, the *performance* evaluator, estimates the mass that the launcher chosen by the user is capable of putting into the required orbit. The second, the *fairing* evaluator, checks if the

satellite (when all the components are folded), fits into the space designed to it in the launcher system. To accomplish this, the model uses a database of launcher with information on its fairing dimensions, and performance capabilities.

Inputs and Outputs

Table 2.8 shows the inputs and outputs of the launcher model, and each of them is explained below the table:

Input Parameters	Output Parameters
Orbit Parameters	Launcher Performance
Satellite Wet Mass [kg]	Fairing fitting
Satellite Dimensions [m]	
Launcher System	

TABLE 2.8: Launcher subsystem Table.

- *Satellite Wet Mass*: The mass of the spacecraft at launch.
- *Satellite Dimensions*: Dimensions of the satellite in the following reference frame: the length corresponds to the vertical longitude of the satellite along the direction of launch, and the diameter to the maximum transversal distance inside the fairing, when the satellite has all its components folded.
- *Launcher system*: The model contains a list of launchers and information of their performance (weight that they can put into orbit) as function of the type of orbit and altitude. Information about the dimensions of the fairing is also provided. The launchers that the system supports are: Ariane5 ESCA, Soyuz, Vega, Delta 7320, Delta 7420, Delta 7920, Taurus XL and Minotaur IV.
- *Launcher performance*: Mass that can be put into a specific type of orbit and altitude.
- *Fairing fitting*: The model checks if the length and the maximum horizontal distance, when the satellite has all its components folded, are smaller than the fairing dimensions.

Description of the model

The following part describes the sizing estimation method used in the launcher subsystem, explaining in detail all the different models and equations used all along

the process. All the equations and estimates used are explained in detail in chapter 18 of the SMAD book [43].

In this specific subsystem, the rule based model will check if the satellite dimensions fit into the launcher system selected by the user and check if the launcher is capable of putting the satellite's wet mass into the desired orbit. So basicly, the system will look up the diameter and height of the launcher in a database and check if the satellite's dimensions are smaller than those limits. If they aren't the system will recommend to look for another launcher.

Secondly, for the performance, the launcher database contains the coefficients of second degree polynomial that estimates the mass that the satellite is capable of putting into a specific type of orbit and altitude. For example, for the Delta 7320 launcher system, the polynomial for LEO is:

$$Mass = 2187h^2 - 0.53125h + 4.06 \cdot 10^{-5}$$

where h is the altitude of the orbit. The information of this database is obtained from VASSAR [36].

Recommendations

Once all the parameters are calculated, the system will compare the following values between the CHECK fact (information given by the user) and the DESIGNED fact (information calculated by the system), and give different recommendations depending on which of them is higher or if they are equal, within an error of 10%:

- **Launcher performance:** check if the launcher has the capability to put the mass of the satellite into the required orbit altitude.
- **Fairing dimensions:** check if the satellite (when folded down) fits into the fairing of the satellite.

2.3 Front-end Web Interface

As it was described in the overview of this chapter, the Satellite Design Assistant has a web interface to act as a front-end for the user. In this section, the different parts and functionalities of the website will be explained. The server-side of the web interface is also considered to be in this part, as it works independently of the

back-end rule-based expert system. Only when the user wants to evaluate a specific subsystem, the server side of the webpage calls the backend of the system.

The back-end of the webpage was developed using the Spring Framework [32]. Spring makes it easy to create Java enterprise applications. It provides everything you need to embrace the Java language in an enterprise environment, with support for Groovy and Kotlin as alternative languages on the JVM, and with the flexibility to create many kinds of architectures depending on an application's needs. Spring manages all the dependencies required by the project, and made easier to connect the server side of the webpage with our rule-based expert system written in Jess, which is written in Java. On the other side, all the front end has been developed using HTML, CSS (Bootsratp library [6]), and JavaScript (Jquery library [41]).

The next sections explain the different funcionalities of the webpage, starting with the ways to input the user's design information and editing it, and finishing with the report page where the recommendations are displayed.

2.3.1 Voice/ Text Input

Figure 2.4 shows the main ways to input information into the system. Specifically, this section is going to explain the first input tag, at the top of the webpage. The user can write any sentence explaining the design, or use the voice recognition button to speak to it.

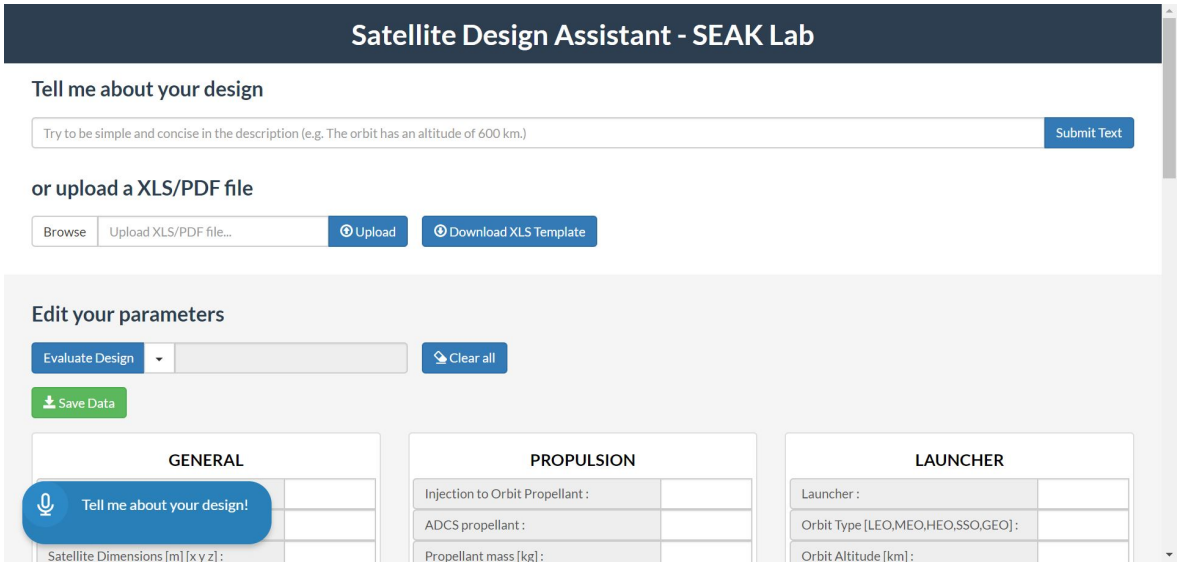

FIGURE 2.4: View of the top of the webpage.

If the user decides to write a sentence explaining a specific parameter (i.e. *The satellite has a dry mass of 200 kg.*), the text will be sent to a python script which uses

the Natural Language Processing (NLP) library Spacy [12], to extract any parameter of the database and save it. The user can also talk to the webpage, by activating the voice recognition tool. In this case, the webpage will transform the speech into text using the SpeechKit library [**?**], which writes what the user has said into the input bar. Then, the user can edit the text if it was not correctly interpreted and submit it, calling again the same python script which will detect any parameters and their values.

The python script has been designed assuming sentences will have simple structures and follows these steps to extract the parameter names and their values:

1. **Syntax detector:** Firstly, a function divides the text into sentences, in case there is more than one sentence as an input and process them separately. Then, the script assigns a *syntac type* to each sentence, meaning that it will detect if there is a conjunction in it or is a simple sentence. If the sentence is a conjunction, the script will look for 2 or 3 parameters inside a same sentence and, on the other hand, if the sentence has a simpler structure (e.g. *The EPS mass is 70 kg* or *The altitude of the orbit is 600 km.*), the script will only look for one parameter.

2. **Parameter detector:** Secondly, for each function, the library used (Scapy), lets you detect the parts of speech in a sentence, whether a word is a verb or a noun and their function in the sentence, whether its the subject or the object. Then, it look for the subject of the sentence and attaches adjectives and nouns that complement the subject to build the parameter name. In the case that there is a conjunction and, as consequence, more than one parameter, the script will look for the subjects of the subsentences in the conjunction.

3. **Value detector:** Then, for each parameter detected, the script will look for an attribute of that subject, that can be either numerical, like in most cases, or non-numerical, for some of the parameters.

4. **Assigning a parameter from the Database:** The Difflib python library is used to compare the *string* detected by the python module with the real parameter names on the database. The parameter that results more similar to the words detected by the python script, will have assigned the detected value and saved into the database. Figure 2.5 showw the dashboard of the webpage, where values saved in the database are shown in bold, when it detects the parameters of the sentence: *The satellite has a dry mass of 200 kg and an orbit altitude of 600 km.*

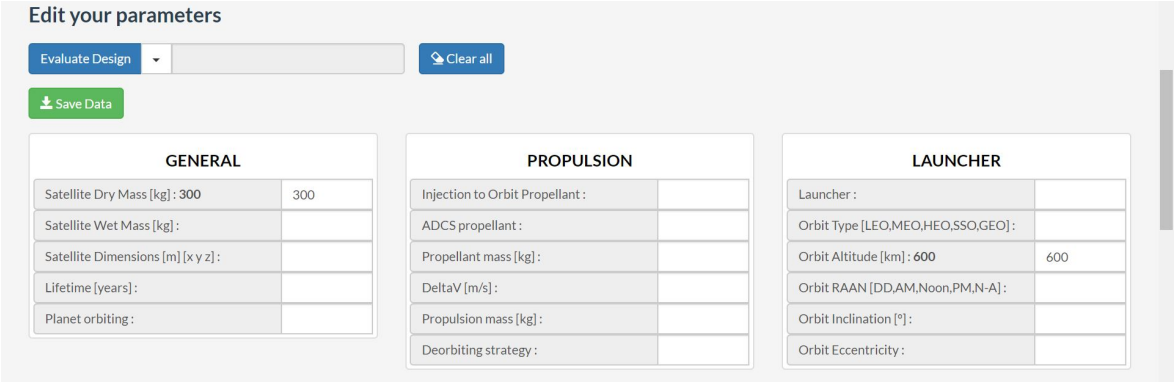

FIGURE 2.5: Parameters that are detected are directly saved into the database and shown in the dashboard.

5. **Message:** Lastly, a message is displayed in the webpage with the parameter and value detected, or a message saying that no parameters were detected if it's the case. Figure 2.6 shows an example of the message displayed in the webpage:

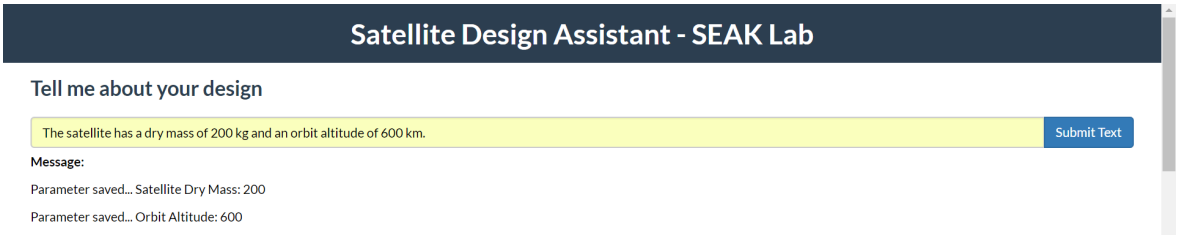

FIGURE 2.6: Output message when parameters are saved.

2.3.2 File Input

Another option to input information into the system is by uploading files. This option was implemented because text and voice recognition work better by detecting parameters one by one in simple sentences. If the user wants to upload a full design with most of the parameters at once, it is more efficient to do it through file uploading. Figure 2.7 shows the view of this part of the webpage:

FIGURE 2.7: File upload section of the webpage.

As can be seen, the user can upload Excel Files or PDF files:

- **PDF File:** Many times designs are described in PDF files such as in PDR (Preliminary Design Review) or SRR (Systems Requirements Review). Thus, it was important to implement a tool like this one, where PDF files can be read to extract the relevant parameters. At this point, the tool is not very accurate because it only reads the raw text of the PDF, divide it into sentences and send the sentences to the python script which detects the parameters from the text input described in section 2.3.1. Therefore, the tool works better when text is preprocessed and cleaned, but it can be a first step to a further development of this useful tool.
- **Excel File:** The simplest and most effective option to upload parameters is through an Excel File. The user can download a template with a list of all the parameters in the database and fill it with its design. By doing so, the user can be sure that the parameters will be correctly detected, and there won't be detection errors like in the case of PDF files.

2.3.3 Editing Dashboard

The dashboard is the main part of the webpage. There, all the parameters in the database are listed in their respective subsystem, plus an extra list of general information that is common to all the subsystems. When a value is saved into a specific parameter of the database, it appears next to its respective parameter showing the user which of them are empty. Figures 2.8 and 2.9 show a split view of all the dashboard.

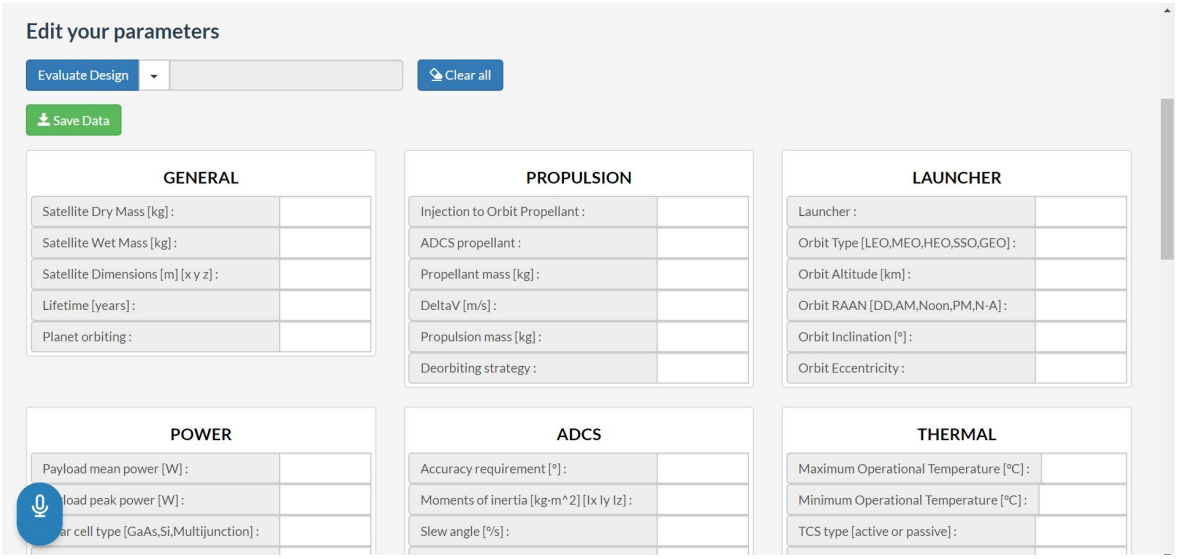

FIGURE 2.8: Top view of the dashboard to edit parameters in the webpage.

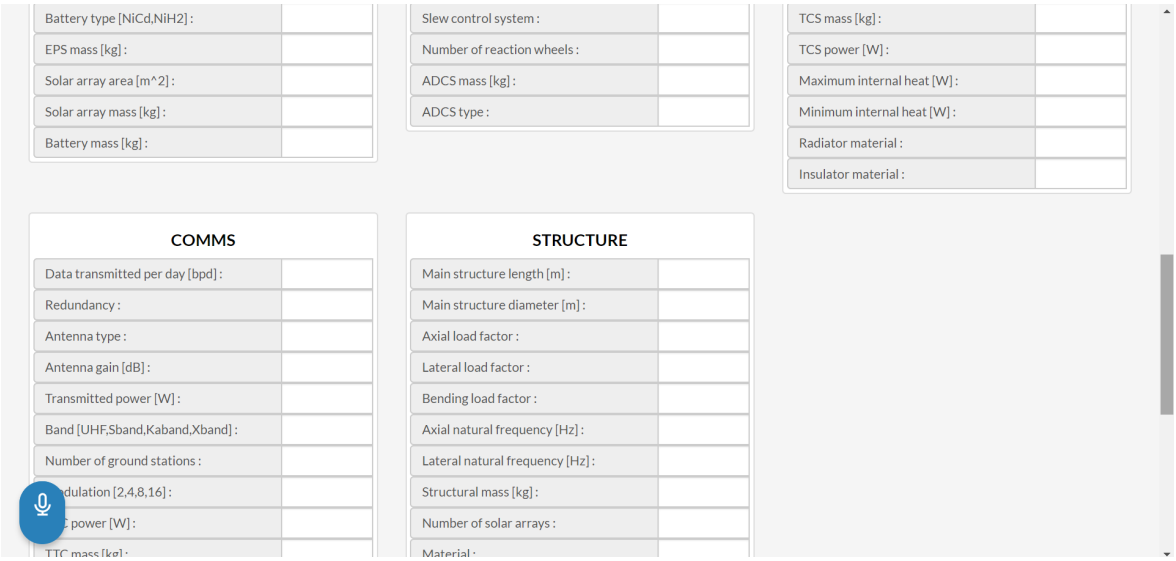

FIGURE 2.9: Bottom view of the dashboard to edit parameters in the webpage.

The functionalities of the dasboard are:

- **Parameter editing:** The main function of the dashboard is showing the values of the parameters of the database. It may happen that the text recognition module, incorrectly reads a sentence and saves a wrong value for some parameter. In the dashboard, the user can see the error and edit it, by writing the correct value for a parameter and then clicking the **save** button, which updates the parameter database with the information written in the dashboard. Figure **??** shows an example of how the webpage looks when parameters are saved into the database.

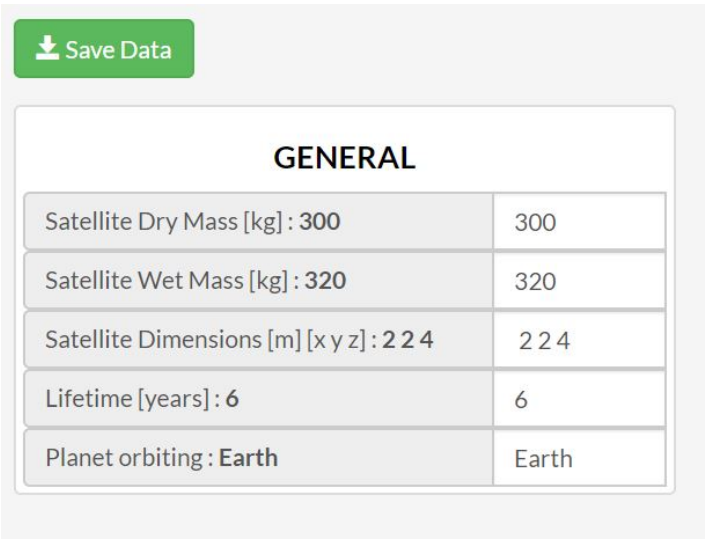

FIGURE 2.10: View of some parameters being saved into the database.

- **Clear All:** This button at the top of the dashboard, deletes all the values of the parameters in the database to erase all present information.
- **Evaluate Design:** This button is used when all the parameters have their values saved and the user wants to evaluate a specific subsystem, or all at once if its the case. If a parameter is empty and is required for the subsystem being evaluated, a message will appear saying that there is some missing information. As shown in Figure 2.11, the user can select which subsystem to evaluate and the selection will appear next to the button:

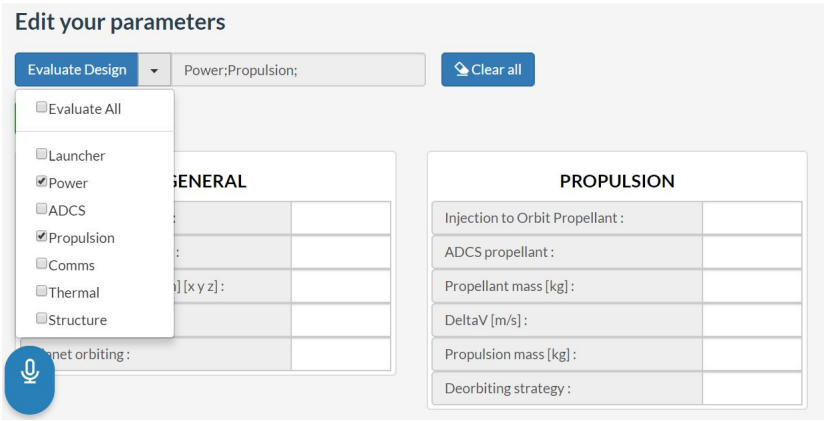

FIGURE 2.11: View of the dropdown menu to evaluate a specific subsystem.

As mentioned in this section, when a user clicks the evaluate button for a specific subsystem, the server side of the webpage collects all the parameters required to evaluate the subsystem in the rule-based expert system, and sends them to the back-end of the system. This is an important step, because there can be examples where a subsystem needs information of other subsystems; for example, to evaluate the propulsion subsystem, the model also needs information on the ADCS control type, but in the webpage, the user doesn't see them in the same list but in different blocks.

To solve this possible problem, the server side of the webpage knows the parameters required to evaluate each model, that are different from the simplified lists showed in the webpage. If a parameter is missing, the webpage will show a message to the user, as shown in Figure 2.12:

Lastly, at the bottom of the dashboard, the user can find a guide to help with the inputs that can be put into the model. Some parameters only allow a discrete number of options that are listed in this help part:

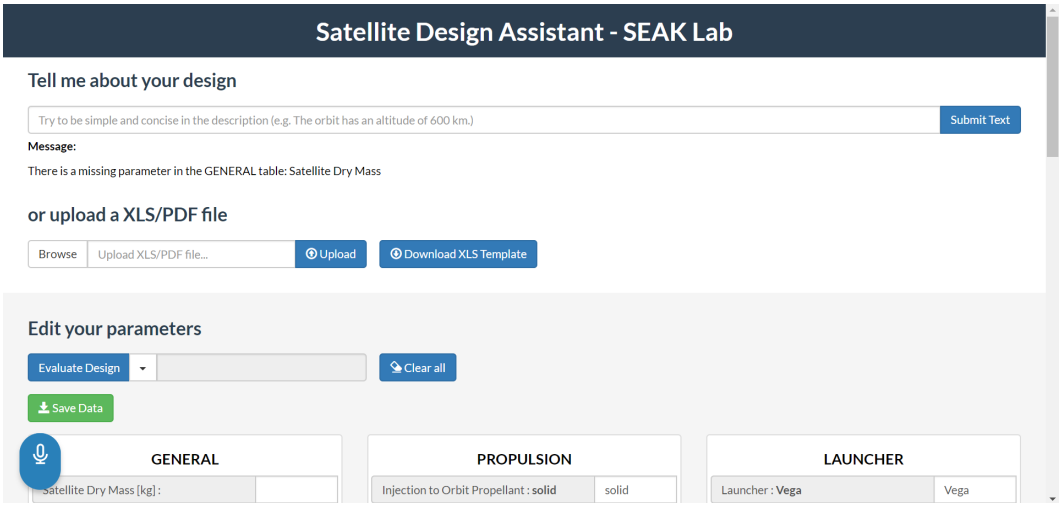

FIGURE 2.12: View of the message displayed when there is a required parameter empty.

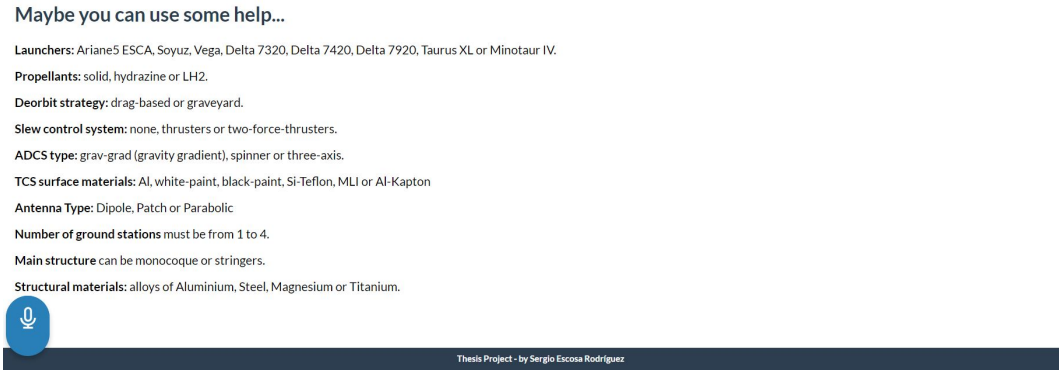

FIGURE 2.13: Guide to help the user to use the system, at the bottom of the page.

2.3.4 Report

The report is a new page where all the recommendations are shown to the user. When one or more subsystems are evaluated by the back-end, it sends a list of recommendations for each of the subsystems, and they are rendered in the page with a small icon next to them to differentiate the positive from the negative ones, as shown in Figures 2.14, 2.15 and 2.16 [1]:

[1] this example was only to show a view of the report page, the numbers may not be coherent.

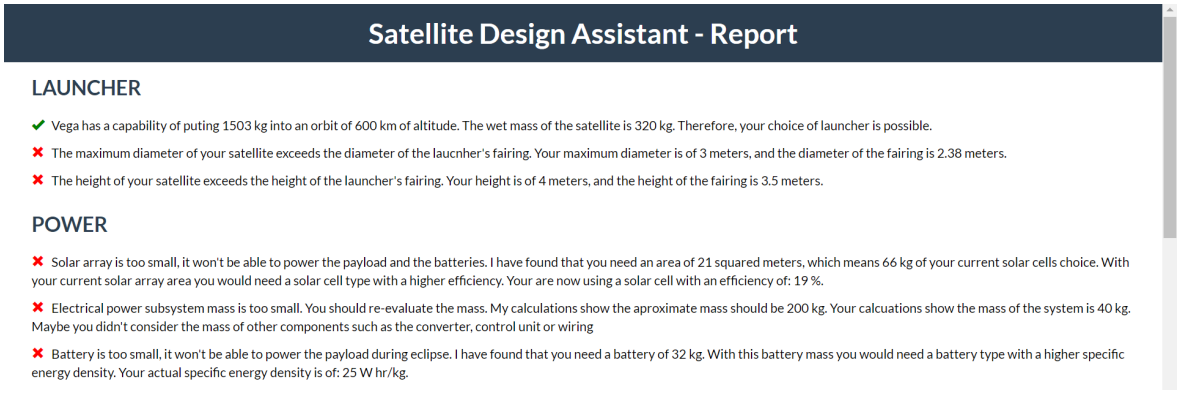

FIGURE 2.14: View of the report page: Launcher and Power subsystems.

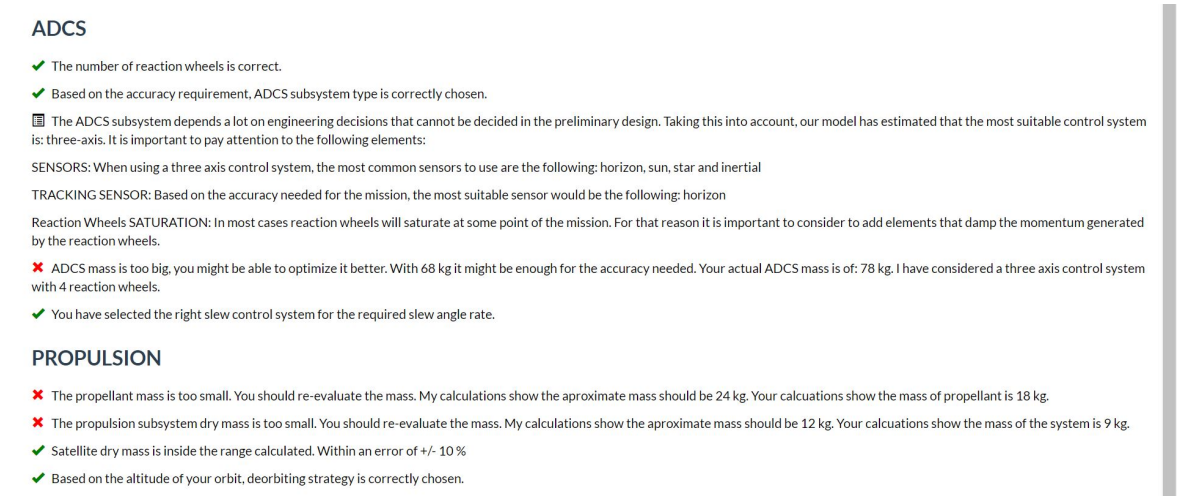

FIGURE 2.15: View of the report page: ADCS and Propulsion subsystems.

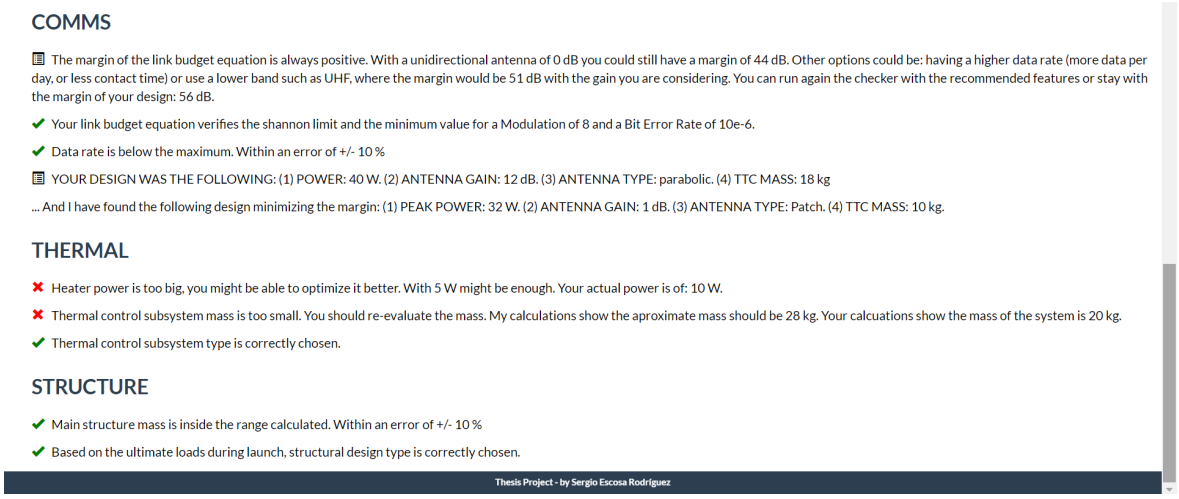

FIGURE 2.16: View of the report page: Communications, Thermal and Structures subsystems.

Chapter 3

Use Case

This chapter explains the possible use cases where the described tool could be used in a practical way. The first section shows how the satellite design assistant could be used in a concurrent design facility (CDF), and the second section explains how professors could use the tool to automatically grade projects from students. This section also includes two examples where the assistant was used to receive feedback of two different satellites: a CubeSat, which was designed by students in a class project and FireSat, a real satellite that uses the SMAD book as example [43].

3.1 Concurrent Design Facilities

Nowadays, large space organizations such as NASA and ESA have design facilities where engineers speed up the mission design process by rapidly assessing the cost, value, feasibility and risk of multiple mission concepts, as explained in [4]. Furthermore, when the mission concept is fixed, it is a work methodology used to speed up the design process by working with all the subsystem engineers together in one room. This work methodology requires to gather all the information of the different subsystems in one place and put in common many design decisions that affect the overall performance of the satellite.

The satellite design assistant described in this project could be a useful tool to work with in these design sessions. All the high-level information of the spacecraft would be in one place and it could be introduced into the system in many ways (voice, text, files, or edit manually). Then, engineers could iterate the design in real time and receive high-level recommendations of the current option they are working with. Another advantage of working with the tool would be that the design could be shared to all participants and they could edit the dashboard in real time, making it more interactive for a group meeting.

Having this global view of the design is a great advantage during preliminary phases, and would make it easier to iterate the design. Once the recommendations of a specific design are received, the engineers could change the parameters that are not correct in real time and iterate to obtain an optimal global design. At this moment the tool doesn't iterate the design automatically, but it can be done manually by changing the inputs when a negative recommendation is received.

As it will be explained in section 4.2, some future work would have to be done to make the assistant more shareable with different users, such as a user authentication feature, where each user can save its design or share it with the concurrent design session.

3.2 Automatic grading

Probably all aerospace students have done a project where they have to design a space mission based on a list of requirements. When professors have to grade this type of projects, it can be difficult to pay attention to all the calculations and design parameters and see if they are correct. This tool could be useful in this situation. The professor would input the parameters of the student's design, run the assistant and receive the feedback. Then it would be easier to pay attention in the negative recommendations.

In order to test this use case, the director of this thesis gave me an example of Preliminary Design Review (PDR) that his students had done in a class project. To start with the test, it was firstly tried to input the PDF document directly into the webpage, but nothing was correctly read by the system. The parameter extractor only works accurately with simple and concise sentences, and that could be something to work more in the future, because the capacity of reading PDR documents and extracting parameters would be very useful. A second try was done by preprocessing the PDF file and cleaning it, leaving only the parts where parameters could be read. This option had better results and some of the parameters were read by the system. The incorrect or missing ones were manually written into the dashboard.

This example consisted of a CubeSat of 15 kg in LEO, and when the system evaluated the model only 40% of the recommendations were positive meaning that those were exactly the same as the ones calculated by the model. This could be by many reasons, of course one is that the physics of the model are far from being accurate, but more importantly, the model uses many empirical relations taken from the SMAD book and other sources. These empirical relations are obtained from historical databases

of large monolithic satellites built in the early years of space explorations. Thus, they might not be applied to modern small satellites such as this CubeSat. To further confirm this hypothesis, it was noted that the subsystems that had more empirical relations such as the ADCS or the thermal subsystem had more negative recommendations that others that depend more on first order physics equations, such as the communications or the power subsystem.

To further test the system, another example was evaluated by the assistant. The FireSat satellite is an example that the SMAD book uses constantly in all the subsystems. It is a 250 kg satellite in LEO. In this case, there were less errors and more positive recommendations, nearly 80% of the recommendations were positive. Of course, for a real satellite the system would have to find no mistakes ideally, and give 100% of positive feedback. However, there is an improvement of the system's accuracy when evaluating large and real satellites than when evaluating a cubesat designed by a group of undergraduate students.

Chapter 4

Conclusion

4.1 Limitations

Some difficult to solve problems identified during the process were identifies and are listed as limitations of the project:

- **Small testing:** There has been very few formal testing with humans to see if the system really accomplishes its intended benefits. The only evidence of its accuracy in recommendations are the examples shown in chapter 3 with the two examples of satellites.
- **Generalization:** The system is not thought out to be used in any kind of spacecraft. As it was shown in the use case, the empirical relations used in the model can only apply to large satellites that orbit the Earth. A future task could be developing an interplanetary model that can design any spacecraft in the solar system.
- **Execution time:** Some of the subsystems require the execution of Matlab files that are executed from a Java environment. This task makes the execution time longer only because of the time it takes to connect to Matlab. This could be solved by writing the whole code in Java, giving up the benefits of some of the Matlab libraries used.

4.2 Future Work

While developing the whole system, a lot of ideas were postponed as they were considered not to be basic for the project to actually be in a working state:

- **More granularity in the recommendations:** Making the recommendations more explainable has been a difficult task during the project, and one option to make them more useful to the user would be having more depth in the feedback, meaning that if a negative recommendation is shown, the system could be able of finding the mistake at a lower level of design. This is done in some of the subsystems but I think it could be done going even further to showing more detailed recommendations.
- **Global optimization of the design:** At this moment, if the user wants to optimize the design globally it has to be done manually by editing the inputs with the recommendations received in the previous iteration. Due to a lack of time, this wasn't done automatically, but it would be great improvement to have an iterator that finds an optimal solution.
- **Interplanetary model:** The model is only thought to design Earth orbiting missions, and having models of other planets would be a great complement to the current system. This task was started with some of the subsystems, but was not finished due to a lack of time.
- **User authentication:** This feature would be very important for the concurrent design facilities use case, where different users could log in and save their different designs or share them with other users.

4.3 Summary

The aim of this final degree thesis was to develop a full functional cognitive assistant for helping system engineers in the high-level design of spacecrafts. This main objective has been fully completed with the development of the whole architecture of the satellite design assistant, including the front-end and the back-end of the system.

Until some years ago, most cognitive assistants have been developed for commercial general usage, then the Systems Engineering, Architecture and Knowledge Lab at Texas A&M University developed *Daphne*, the first cognitive assistant specialized in space mission design. The current thesis, has explained a possible complement of *Daphne* to develop the individual satellites of a space mission. Ideally, one would start designing the whole mission architecture with *Daphne*, and fix the payload and orbits of each satellite, and then, start with the preliminary design of each spacecraft with the help of the Satellite Design Assistant described in this project.

The problem of cognitive overload of humans is becoming common in a lot of fields, with design and aerospace being two of them, and there is a need for tools that assist humans in these complex tasks. I find important to explain here, that these kind of assistants are thought and developed to be tools that the human can use at any time to help him in difficult tasks, and are not thought to replace engineers or to do all the work by themselves.

On a more personal note, this project has been a great challenge for me, both for the amount of coding and technical knowledge that I had to learn during the process. I am specially proud of having developed an end to end software architecture for the first time, without any past experience in either web development or languages such as Java and Jess that are the basis of the back-end module. This experience has been very valuable for my future work experience because nowadays, technologies change very rapidly and one could be working in a totally new field in a few years. It is the capacity of rapidly learning new things what I value most of my time at the university, and what I personally think that is more valuable than any technical knowledge we learn during these years.

Last but not least, finishing this project and thesis brings a sense of closure for me, after 5 years of hard work doing two bachelor degrees at the same time, this is the end result of the whole effort. I could not be more prouder of it, as I have been able to use a lot of the knowledge I learned along the way in both of my degrees, and prove to myself that I could take a big challenge like this one.

Appendix A

Source code of the project

The source code for the whole project can be found under different repositories in Github, under the `seakers` group: https://www.github.com/seakers. The ones that are mentioned during the project are listed here, the **Satellite Design Assistant** corresponds to the source code of this thesis and the others show the work done by the SEAK Lab with Daphne and VASSAR:

- **Satellite Design Assistant**: https://github.com/seakers/satellite-design-assistant
- **Daphne Web page**: https://www.selva-research.com/daphne/
- **Daphne Brain**: https://github.com/seakers/daphne_brain
- **VASSAR**: https://github.com/seakers/VASSAR
- **Daphne Interface**: https://github.com/seakers/daphne-interface

References

[1] Kuhne electronic - Microwave components. URL https://www.kuhne-electronic.de/.

[2] Amazon,. Amazon Alexa. URL https://developer.amazon.com/es/alexa.

[3] Apple,. Siri - Apple. URL https://www.apple.com/siri/.

[4] Bandecchi, M., Melton, B., Gardini, B., Estec, E. S. A., Noordwijk, N.-A. G., and Ongaro, F. The ESA / ESTEC Concurrent Design Facility. (January 2000), 2015.

[5] Bang, H., Virós Martin, A., Prat, A., and Selva, D. Daphne: An Intelligent Assistant for Architecting Earth Observing Satellite Systems. pages 1–9, 2018. doi:10.2514/6.2018-1366.

[6] Bootstrap,. Bootstrap - The most popular HTML, CSS, and JS library in the world. URL https://getbootstrap.com/.

[7] Buchanan, B. G. and Shortliffe, E. H. *Rule-based expert systems : the MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.

[8] CS Systèmes d'Information,. Orekit Library. URL https://www.orekit.org/.

[9] Donath, D., Rauschert, A., and Schulte, A. Cognitive assistant system concept for multi-UAV guidance using human operator behaviour models Cognitive assistant system concept for multi-UAV guidance using human operator behaviour models. *Humous'10*, (November), 2010.

[10] Durkin, J. *Expert systems : design and development*. Macmillan, 1994.

[11] Engelbart, D. C. *Augmenting Human Intellect: A Conceptual Framework*. 1962. URL https://www.dougengelbart.org/pubs/papers/scanned/Doug{_}Engelbart-AugmentingHumanIntellect.pdf.

[12] Explosion AI,. spaCy · Industrial-strength Natural Language Processing in Python. URL https://spacy.io/.

[13] Ferrucci, D. A. Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3.4):1:1–1:15, may 2012. doi:10.1147/JRD.2012.2184356.

[14] Forgy, C. L. Rete: A Fast Algorithm for the Many PatternIMany Object Pattern Match Problem*. Technical report. URL http://www.csl.sri.com/users/mwfong/Technical/RETEMatchAlgorithm-ForgyOCR.pdf.

[15] Freed, M., Carbonell, J., Gordon, G., Hayes, J., Myers, B., Siewiorek, D., Smith, S., Steinfeld, A., and Tomasic, A. RADAR: A Personal Assistant that Learns to Reduce Email Overload Email and Task Management. Technical report. URL www.aaai.org.

[16] Friedman-Hill, E. *Jess In action*. Number 02. 2003.

[17] Goel, A. K., Vattam, S., Wiltgen, B., and Helms, M. Cognitive, collaborative, conceptual and creative — Four characteristics of the next generation of knowledge-based CAD systems: A study in biologically inspired design. *Computer-Aided Design*, 44(10):879–900, oct 2012. doi:10.1016/J.CAD.2011.03.010.

[18] Google,. Google Assistant, your own personal Google. URL https://assistant.google.com/.

[19] Gross, J. and Rudolph, S. Geometry and simulation modeling in design languages. *Aerospace Science and Technology*, 54:183–191, jul 2016. doi:10.1016/j.ast.2016.03.003.

[20] Guerlain, S. A., Smith, P. J., Obradovich, J. H., Rudmann, S., Strohm, P., Smith, J. W., Svirbely, J., and Sachs, L. Interactive Critiquing as a Form of Decision Support: An Empirical Evaluation. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 41(1):72–89, mar 1999. doi:10.1518/001872099779577363.

[21] Hayes, C. C., Goel, A. K., Tumer, I. Y., Agogino, A. M., and Regli, W. C. Intelligent Support for Product Design: Looking Backward, Looking Forward. *Journal of Computing and Information Science in Engineering*, 11(2):021007, jun 2011. doi:10.1115/1.3593410.

[22] Hoffpauir, D. NASA Systems Engineering Handbook (SP-2016-6105), Rev 2. 2017.

[23] IBM,. IBM Watson Products and Services. URL https://www.ibm.com/watson/products-services/.

[24] Kitamura, Y., Kashiwase, M., Fuse, M., and Mizoguchi, R. Deployment of an ontological framework of functional design knowledge. *Advanced Engineering Informatics*, 18(2):115–127, apr 2004. doi:10.1016/j.aei.2004.09.002.

[25] Maier, M. W. M. W. and Rechtin, E. *The art of systems architecting*. CRC Press, 2009.

[26] McDermott, J. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19(1):39–88, sep 1982. doi:10.1016/0004-3702(82)90021-2.

[27] Microsoft,. Personal Digital Assistant - Cortana Home Assistant - Microsoft. URL `https://www.microsoft.com/en-us/cortana`.

[28] Myers, K., Berry, P., Blythe, J., Conley, K., Gervasio, M., McGuinness, D. L., Morley, D., Pfeffer, A., Pollack, M., and Tambe, M. An Intelligent Personal Assistant for Task and Time Management. *AI Magazine*, 28(2):47–47, jun 2007. doi:10.1609/AIMAG.V28I2.2039.

[29] Nag, S., Hughes, S. P., and Le Moigne, J. Streamlining the Design Tradespace for Earth Imaging Constellations. In *AIAA SPACE 2016*, Reston, Virginia, sep 2016. American Institute of Aeronautics and Astronautics. doi:10.2514/6.2016-5561.

[30] National Aeronautics and Space Administration,. NASA Technology Roadmaps TA 11: Modeling, Simulation, Information Technology, and Processing. Technical report, 2015.

[31] Onken, R. and Walsdorf, A. Assistant systems for aircraft guidance: cognitive man-machine cooperation. *Aerospace Science and Technology*, 5(8):511–520, dec 2001. doi:10.1016/S1270-9638(01)01137-3.

[32] Pivotal,. Spring Framework. URL `https://spring.io/`.

[33] Riley, G., Culbert, C., Savely, R. T., and Lopez, F. N88-16368 CLIPS: An Expert System Tool for Delivery and Training. Technical report. URL `https://ntrs.nasa.gov/search.jsp?R=19880006986`.

[34] Selva, D. Selva Research Group – Systems Engineering Architecture and Knowledge (SEAK-LAB). URL `https://www.selva-research.com/`.

[35] Selva, D. Rule-Based System Architecting of Earth Observation Satellite Systems by. (2004):1–416, 2012. URL `http://systemarchitect.mit.edu/docs/selva12c.pdf`.

[36] Selva, D., Cameron, B. G., and Crawley, E. F. Rule-Based System Architecting of Earth Observing Systems: Earth Science Decadal Survey. *Journal of Spacecraft and Rockets*, 51(5):1505–1521, 2014. doi:10.2514/1.A32656.

[37] Selva, D. and Crawley, E. F. VASSAR: Value assessment of system architectures using rules. In *2013 IEEE Aerospace Conference*, pages 1–21. IEEE, mar 2013. doi:10.1109/AERO.2013.6496936.

[38] Selva, D., Golkar, A., Korobova, O., Cruz, I. L. i., Collopy, P., and de Weck, O. L. Distributed Earth Satellite Systems: What Is Needed to Move Forward? *Journal of Aerospace Information Systems*, 14(8):412–438, jan 2017. doi:10.2514/1.I010497.

[39] Springmann, P. N. and de Weck, O. L. Parametric Scaling Model for Nongeosynchronous Communications Satellites. 2004. URL `http://strategic.mit.edu/docs/2{_}3{_}JSR{_}parametric{_}NGSO.pdf`.

[40] Steele, G. L. *Common LISP*. Elsevier Science, 1990.

[41] The jQuery Foundation,. jQuery. URL `https://jquery.com/`.

[42] Thompson, R. E., Colombi, J. M., Black, J., and Ayres, B. J. Disaggregated Space System Concept Optimization: Model-Based Conceptual Design Methods. *Systems Engineering*, 18(6):549–567, nov 2015. doi:10.1002/sys.21310.

[43] Wertz, J. R., Everett, D. F., and Puschell, J. J. *Space mission engineering : the new SMAD*. Microcosm Press, 2011.

[44] Wolphram Alpha,. Wolfram | Alpha: Computational Knowledge Engine., 2019. URL `https://www.wolframalpha.com/`.