POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Aerospaziale

Tesi di Laurea Magistrale

Architecture definition of a Thermal Management System for a hybrid-electric aircraft with a MBSE approach

Relatori

Dr. Davide Ferretto

Ing. Ricardo Reis

Dr. Roberta Fusaro

Prof. Nicole Viola

Candidato

Davide Vertemati

Aprile 2023

Abstract

This study involves the analysis of the conceptual design of a thermal management system designed for a hybrid-electric aircraft, with the objective of establishing a digital and seamless connection between the system model, developed using Model Based System Engineering, and numerical simulation models. The system overview is provided by Embraer Research and Technology Europe - Airholding S.A, the project partner, which also supplies the Matlab and Simulink numerical models utilized throughout the study. The first step is to conduct an evaluation of the engineering workflow to identify areas for improvement where the above-mentioned connection could enhance the system development process. The subsequent focus of the study is on creating the tool required to establish the seamless connection, which comprises of an MBSE architectural model built using the ARCADIA methodology in Capella software, and a digital bridge developed in Python programming language, linking the MBSE model with the Matlab and Simulink models. The outcome is a shareable MBSE TMS formal model, functioning as an accessible framework for all the engineers involved in the design process, this contains the system characterization and is directly connected to its corresponding numerical model via the digital bridge.

List of contents

List of Tables

List of Figures

List of Formulas

Abbreviations

MBSE	Model Based System Engineering
MBD	Model Based Design
TMS	Thermal Management System
INCOSE	International Council on Systems Engineering
ARCADIA	Architecture Analysis & Design Integrated Approach
PMTE	Process Method Tool Environment
EPBS	End Product Breakdown Structure
V&V	Verification and Validation
CNL	Controlled Natural Languages
HEA	Hybrid-Electric Aircraft
TLARs	Top-level Aircraft requirements
CS	Certification Specification
ICAO	International Civil Aviation Organization
SL	Sea level
ISA	International Standard Atmosphere
MTOM	Maximum Take-Off Mass
SAF	Sustainable Aviation Fuel
Cr	Chord Root
Ct	Chord Tip
Cma	Aerodynamic Medium Chord
EM	Electric Motors
GT	Gas-Turbine
M/G	Motor/Generator
PMDU	Power Monitoring and Distribution Unit
FMDU	Fuel Monitoring and Distribution Unit
LCC	Liquid Cooling Cycle
VCS	Vapour Cooling System
EGW	Ethylene-Glycol
SHX	Skin Heat Exchanger
RAHX	Ram Air Heat Exchanger
MDOA	Multi-Domain Optimization Analysis

PS	Propulsion System
ES	Energy Storage
PT	Power Transmission
PVMT	Property Value Management Tool
P4C	Python4Capella
TBD	To Be Decided
COP	Coefficient of Performance
EPS	Electrical Power System
ECS	Environmental Control System
IPS	Ice protection system
FCS	Flight Control System
SPD	Specific Power Dissipation
MSM	Mode and State Machine
LAB	Logical Architecture Breakdown diagram
PAB	Physical Architecture Breakdown diagram
BOL	Beginning Of Life
EOL	End Of Life
GPU	Ground Power Unit
AI	Artificial Intelligence
ML	Machine Learning
IoT	Internet of Things

1 Context and motivation

The thesis project is concerned with Model Based System Engineering (MBSE), an approach that digitizes Systems Engineering practices. This approach has many advantages over traditional document-based methods, including a single point of reference and the elimination of many of the challenges associated with the latter paradigm.

Aiming to this innovative approach, an interoperable workflow should be established between different tools. These include those that support MBSE and those that model the physical behaviour of systems and their components. The project target is to show how MBSE and Model Based Engineering (MBE) can be linked in a seamless way using the Capella MBSE tool and MATLAB for a Thermal Management System Architecture of a Hybrid-Electric Aircraft. This involves modelling the system in Capella and creating an interlink architecture between the system model and physical model developed in MATLAB.

The MBSE approach offers a solution to some of the complexities and challenges associated with the document-based paradigm. This is achieved by allowing for faster communication between Model Based System Engineering and numerical simulation environments. By exploiting these capabilities, the project wants to boost and make the engineering process more effective.

The intention is that the cross-domain information exchange, which is where most engineering mistakes occur, will benefit from this project. A consistent MBSE model combined with a digital bridge to the simulation environment will hopefully eliminate many of the errors that arise from miscommunication between different domains.

The project aims to demonstrate how MBSE and MBE can be interlinked digitally in a seamless way. By creating a unified digital environment for the system model, engineers will be able to collaborate more efficiently and effectively. This will enable them to identify and resolve issues more quickly, and ultimately, to design and build better systems.

One of the main advantages of the MBSE approach is that it offers a single point of truth reference for the entire system. This means that all stakeholders have access to the same information, reducing the likelihood of errors and misunderstandings. In contrast, document-based paradigms often result in multiple versions of the same information, leading to confusion and errors.

Another objective is to demonstrate the benefits of a digital bridge between the MBSE model and the simulation environment. This will enable engineers to simulate the behaviour of the system in a virtual environment, allowing them to identify and resolve issues before building the physical system. By simulating the system in this way, engineers can reduce the likelihood of errors and improve the efficiency of the design process.

The project focuses specifically on the Thermal Management System Architecture of a Hybrid-Electric Aircraft. This is an important area of research, as hybrid-electric aircraft are becoming increasingly common in the aviation industry. These aircraft have unique design challenges, including the need to manage both electrical and thermal energy. By developing a unified digital environment for the system model, the project aims to help engineers design more efficient and effective hybrid-electric aircraft.

This project is linked to the FutPrint50 effort that in fact regards the development of a Hybrid-Electric aircraft.

Figure 1.1: FutPrint50 logo

This work was preceded by an internship project (30) carried out within the Model Based System Engineering (MBSE) framework, the aim was to carry out a preliminary study of the requirements, tools and strategies for linking the Capella software with programs that simulate the physical behaviour of dynamic systems (e.g. Matlab).

2 Introduction to Model Based System Engineering

Over the past two decades, there has been a significant shift in the engineering and manufacturing sector towards utilizing a model-based approach to system engineering instead of the traditional document-based method. This shift has been driven by the digitization of processes, the need to meet technological demands, and competition among companies to provide higher quality products and services at a faster pace and lower cost (13), (14).

The document-based method involves generating system information and specifications through documents, which can become overwhelming in number for complex projects such as satellite or aircraft design. The large number of documents and the complexity of information can lead to difficulties in management and updating, which can result in cross domain communication problems (15), (16). Additionally, sometimes documents may contain conflicting information, which can lead to unexpected outcomes (15).

In contrast, model-based system engineering (MBSE) involves the formal use of modelling to support system requirements, design, analysis, verification, and validation activities from the conceptual phase throughout development and later life cycle phases (15). According to E. Burger's doctoral thesis "A conceptual MBSE framework for satellite AIT planning," MBSE uses coherent, integrated, and complete computerized models (17) of physical systems and processes (18) that represent an abstract representation of reality (16).

MBSE brings numerous benefits to the engineering and manufacturing sector. Improved communication between designers and customers is facilitated by better data sharing and information exchange (15), (16). Enhanced management of system complexity and reduced risk of limitations are also benefits of MBSE (15), (16), (1). Higher product quality is achieved through a unique system model (15), while increased productivity is achieved through reduced time and cost with the help of recycled models and traceable requirements (15), (1). Additionally, standardized models make it easier to teach and learn MBSE concepts (15).

One of the primary advantages of MBSE is improved communication between designers and customers. The use of a coherent and integrated computerized model facilitates better data sharing and information exchange between stakeholders. This improved communication leads to a better understanding of the system's requirements and enables designers to respond to customer needs more efficiently (15), (16).

Another significant benefit of MBSE is enhanced management of system complexity and reduced risk of limitations. The use of a model allows for the identification and management of interdependencies between system components. This, in turn, helps to reduce the risk of limitations and ensures that the system functions as intended (15), (16), (1).

Higher product quality is also a plus of MBSE, the use of a unique system model ensures that all stakeholders have a common understanding of the system's requirements, design, and functionality. This results in a more cohesive and robust system that meets customer needs and specifications (15).

The productivity is increased as MBS allows reduced design time and cost, the use of recycled models and traceable requirements helps to reduce the time and cost associated with system design and development. This, in turn, helps organizations to deliver higher quality products and services at a faster pace and lower cost (15), (1).

Finally, MBSE offers easier teaching and learning of system engineering concepts through standardized models. The use of standardized models helps to ensure that all stakeholders have a common understanding of system engineering concepts, reducing the potential for miscommunication and errors (15).

The shift towards model-based system engineering has been driven by the need to improve the quality of products and services, meet technological demands, and compete in a rapidly changing engineering and manufacturing sector. MBSE offers all the numerous benefits mentioned above: improved communication, enhanced management of system complexity, higher product quality, increased productivity, and easier teaching and learning of system engineering concepts. As organizations continue to adopt MBSE, it is expected to become the standard approach to system engineering in the future.

While Model-Based Systems Engineering (MBSE) offers numerous benefits, implementing it can also present various challenges for organizations in the engineering and manufacturing sector. One challenge is the need for a significant cultural shift within the organization to move from a document-based approach to a model-based one. This shift requires rethinking traditional practices and adopting new methods and tools, which can be challenging and require extensive training for the team.

Another challenge is the potential complexity of the models themselves. As the number of components and subsystems in a system increases, the models can become intricate and difficult to manage. It requires careful planning, design, and organization to ensure that the models remain understandable, maintainable, and scalable.

Furthermore, integrating MBSE into existing systems and processes can be challenging, particularly when those systems and processes have been in place for a long time. Integrating MBSE into existing processes requires careful coordination and communication, and organizations may need to invest in additional technology or personnel to make the transition smoothly.

Finally, there is the challenge of ensuring that the models accurately reflect reality. The models must accurately represent the system and its behaviour, and any changes to the system must be reflected in the model. This requires careful coordination between stakeholders and a well-defined change management process.

MBSE implementation challenges are in conclusion cultural shift, model complexity, integration into existing systems, and ensuring that models accurately reflect reality. Addressing these challenges requires careful planning, coordination, and communication, and organizations must be willing to invest in the necessary training, technology, and personnel to make the transition to MBSE successfully.

As Model-Based Systems Engineering (MBSE) continues to gain widespread adoption across various industries, several potential future developments could emerge. One such development is the increasing use of Artificial Intelligence (AI) and Machine Learning (ML) techniques to analyse system models and automatically generate them. Additionally, the use of Virtual and Augmented Reality technologies could provide more immersive and interactive experiences for MBSE users. Another development could be the integration of MBSE with the Internet of Things (IoT), enabling the creation of more complex systems that can adapt and learn from real-time data. Finally, the standardization of MBSE methodologies and tools across different industries could help to further accelerate its adoption and enable more seamless collaboration between organizations.

2.1 Definition of methodologies, processes, tools

Implementing MBSE requires the application of the appropriate methodology, consisting of "processes," "methods," and "tools." According to J. Estefan's "Survey of Model-Based Systems Engineering (MBSE) Methodologies, 2008," a process is a logical sequence of tasks performed to achieve a particular objective. It defines 'WHAT' is to be done but does not specify 'HOW' each task is performed. The structure of a process provides multiple levels of aggregation, allowing analysis and definition at different levels of detail to support various decision-making needs. On the other hand, a method consists of techniques for performing a task and defines the 'HOW' of each task. At any level, process tasks are performed using methods. But each method is also a process, with its sequence of tasks to be performed. The 'HOW' at one level of abstraction becomes the 'WHAT' at the next lower level.

A tool is an instrument that enhances the efficiency of a task when applied to a particular method. It facilitates the accomplishment of the 'HOWs.' In broader terms, it enhances the 'WHAT' and the 'HOW.' MBSE is primarily based on tools, which are usually software, that allow for the creation of models using appropriate languages. There is a wide variety of tools available on the market, and designers can choose the one that best suits their needs based on the strengths and weaknesses of the software. It is not possible to adopt MBSE without a specific tool.

The methodology of MBSE is defined as the set of processes, methods, and tools that allow for the application of model-based techniques in different project phases. It is a "recipe" that can be applied to problems that have common elements. J. Estefan notes that the concept of "environment" is also related to these definitions. The environment is everything that surrounds us, including external objects, conditions (such as social, cultural, personal, physical, and political), and factors that can influence the choices of a community.

Therefore, the MBSE environment must allow for the integration of the tools and methods used in a project, enabling both the 'WHAT' and 'HOW.' Figure 3.2 shows a visual diagram of the PMTE elements (process, methods, tools, environment) and their links with technology and people, which should always be considered in every project. This emphasizes the importance of choosing the right tools and methods that align with the specific project's requirements and goals.

In conclusion, the implementation of MBSE requires the application of the appropriate methodology, consisting of processes, methods, and tools. A clear definition of these terminologies is crucial to avoid ambiguities and semantic discrepancies between academia and industry. Choosing the right tools and methods that align with a specific project's requirements and goals is critical in the MBSE environment.

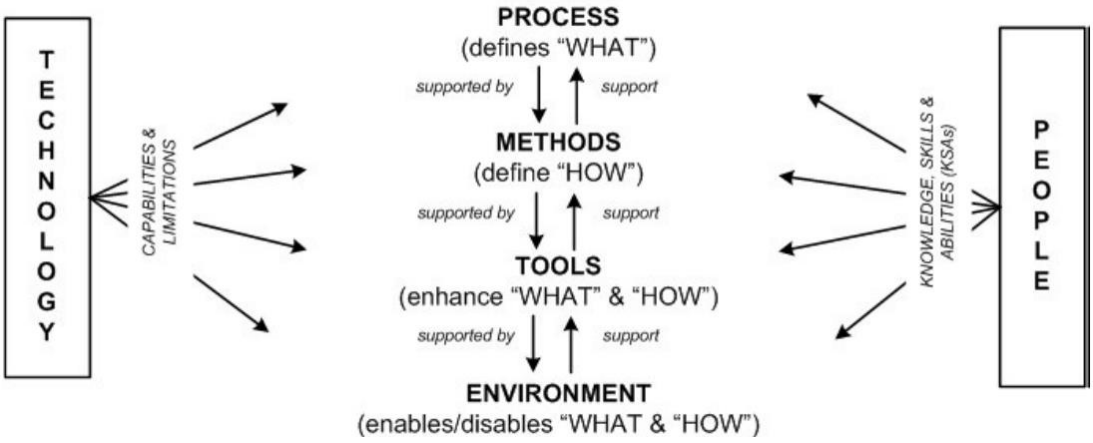

Figure 2.1: The PMTE Elements and Effects of Technology and People (20)

2.2 The ARCADIA methodology

ARCADIA, which stands for Architecture Analysis & Design Integrated Approach, is a methodology developed by Thales Airborne Systems in 2007 with the aim of defining and validating the architecture of complex multidomain systems (1). The ARCADIA methodology is designed to be flexible and can be used with top-down, bottom-up, or middle-out approaches, allowing stakeholders to share information and engage in co-engineering by jointly creating models that are not only descriptive but also interact with each other (1).

ARCADIA is used by Thales for complex systems across various industries, including Defence, Space, Aeronautics, Land Transport, and Security (1). The methodology covers all project activities at different levels, allowing stakeholders to share information and engage in co-engineering by jointly creating models that are not only descriptive but also interact with each other (1).

The ARCADIA methodology is represented by a series of diagrams with interconnected elements that form the structure and behaviour of the system. The methodology has five work levels (1):

- Operational Analysis: This level identifies the actors and analyses the problems and requests of the users and is the highest level of the methodology (What the user needs to do?)
- System Analysis: This level identifies the functions that the system must perform to meet the user requests (What the system needs to do to meet the user need?).
- Logical Architecture: This level recognizes the logical components of the system, their sub-functions, relationships, and content, considering any constraints (How the system will work?).
- Physical Architecture: This level defines the structure of the system and how it will be executed, identifying the behavioural components that can perform the required functions (How the system will be built?).
- End Product Breakdown Structure (EPBS): This level establishes the design limitations of the architecture and deduces the constraints to be met (Expectations from each component provider).

It is important to note that not all project levels are required, as the Operational Analysis and EPBS levels are considered optional and may or may not be implemented based on design difficulties (1).

The ARCADIA methodology is used to improve communication and collaboration among different stakeholders involved in the design process, from engineers to managers to customers (2). It enables the creation of a model that integrates different domains, such as hardware, software, and human factors, and allows for the identification of potential issues early in the design process, reducing the risk of costly errors later on(2).

In conclusion, the ARCADIA methodology is a flexible and versatile approach to system design and validation that facilitates communication and collaboration among different stakeholders. Its implementation through MBSE tools, such as Capella, enables the creation of descriptive and interactive models that allow for the identification of potential and costly issues.

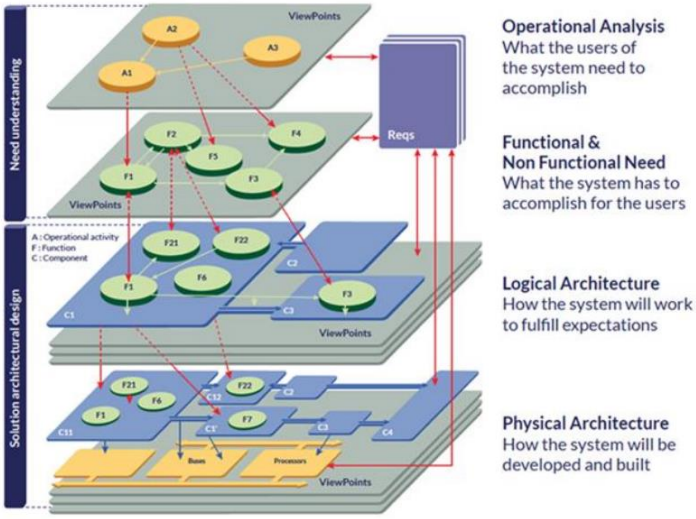

Figure 2.2: The main engineering levels of Arcadia [1]

2.2.1 The Capella software

ARCADIA is accompanied by a modelling tool developed by Thales, which was initially known as Melody Advance and later became open source as Capella. This is a hybrid modelling tool that incorporates elements of SysML and is designed to help with system engineering and architecture problems. Capella provides support for the main levels of the ARCADIA methodology and offers seven types of diagrams for each project stage, including data flow diagrams, scenario diagrams, architecture diagrams, mode and state diagrams, distribution diagrams, class diagrams, and capability diagrams (19), (1). This tool will be used throughout the project, and its diagrams will be utilized in the most suitable manner.

Furthermore, Capella has optional features called add-ons that could be downloaded and used in synergy with the main capabilities, add-ons basically introduce new elements in the MBSE environment, the ones interesting for this project are:

Requirements viewpoint: allow requirement creation and import into the Capella environment so that they can be addressed in the model.

Property Value Management Tool (PVMT): this add-on allows the creation of data, values that could be addressed as properties to various Capella elements.

Python4Capella (P4C): this add-on implements a python interpreter into the Capella environment, this allow to access many of the Capella elements through Python programming language, in this way the language can be used to perform other activities and then change the model on that base.

Capella is a powerful tool that can assist system engineers to model complex systems and optimize their design. In this article, we will explore Capella in more detail, including its capabilities, features, and add-ons. We will also discuss how Capella integrates with other tools to create a complete system engineering solution.

Capabilities of Capella:

Capella is designed to support system engineers at each level of the ARCADIA methodology, from the operational to the logical, and the physical to the implementation level. The tool provides a graphical modelling environment for system modelling and supports the creation of models in various diagrams. Capella's capabilities include the ability to define system requirements, design and architecture, allocate functions, and verify system behaviour.

One of the strengths of Capella is its flexibility in modelling complex systems. For example, Capella can model systems with multiple configurations, or systems that evolve over time. The tool supports the creation of models in different levels of abstraction and allows engineers to describe the system architecture at different levels of granularity. This allows for a more accurate representation of the system and helps to identify potential issues early in the design phase.

Features of Capella:

Capella provides a range of features to support system engineers. One of the key features of Capella is its support for the creation of models in various diagrams. The tool offers seven types of diagrams for each project stage, including data flow diagrams, scenario diagrams, architecture diagrams, mode and state diagrams, distribution diagrams, class diagrams, and capability diagrams (19), (1).

Another important feature of Capella is its support for the creation of system requirements. The tool provides a requirements viewpoint add-on, which allows for the creation and import of requirements into the Capella environment. This feature ensures that requirements are addressed in the model, enabling system engineers to design and optimize the system with the requirements in mind.

Capella also includes a Property Value Management Tool (PVMT) add-on. This feature allows the creation of data and values that can be addressed as properties to various Capella elements. The PVMT add-on is useful for managing properties such as cost, weight, or performance, and can help engineers to optimize the system design.

Furthermore Capella can include the Python4Capella (P4C) add-on, which implements a Python interpreter into the Capella environment. This feature allows engineers to access many of the Capella elements through the Python programming language. This means that engineers can use Python to perform other activities and then change the model on that basis. For example, engineers can use Python to perform simulations, optimization, or data analysis, and then update the Capella model based on the results.

Integration with other tools:

Capella can integrate with other tools to create a complete system engineering solution. For example, Capella can integrate with requirement management tools, such as DOORS, to manage system requirements. In addition, Capella can integrate with version control tools such as Git to manage model versions and changes. This allows engineers to work collaboratively on the same model and track changes made by different team members.

3 Use case description

In this section the FutPrint50 hybrid-electric aircraft is presented, this gives the context within which the TMS will operate.

The partner of this project is Embraer Research and Technology Europe (Airholding), the company provides the use case that is the Thermal Management System for a hybrid-electric aircraft. The system of interest will be described in detail in the following sections, starting from the customer needs and the high-level requirements to the specific TMS layout and the Matlab model used to analyse it.

3.1 The Company

The project has been carried out within Embraer Research and Technology Europe that is the European arm of the Embraer Research and Technology Unit, and is part of Airholding S.A., a full European subsidiary of Embraer S.A.

Embraer S.A. is a Brazilian multinational aerospace manufacturer that produces commercial, military, executive and agricultural aircraft, and provides aeronautical services.

Airholding S.A. is mostly involved in collaborative projects with partners in Europe and elsewhere. The company aims at high level, to bring value to the Embraer group by: fostering strategic relationships with European networks of partners and suppliers, creating, exploiting and sustaining new European based technology and business streams, and contributing to the overall of aviation safety improvement.

In specific, Airholding S.A. is mainly active in projects related with cyber-physical systems, future propulsion, automation, sustainability, and mobility.

Airholding S.A. benefits from Embraer 50-year heritage of successful aircraft design, production, certification and support and more than 14 years collaboration with European partners in Framework and National European R&I programmes. Airholding S.A. can complement its expertise with know-how from the EMBRAER group, extending its capabilities.

Figure 3.1: Embraer logo

3.2 FutPrint50

In recent years, there has been a significant rise in the number of passengers traveling by air, with 4.5 billion passengers recorded in 2019, which is 56% more than in 2004 (24). To address the issue

of harmful emissions that result from aviation, the global aviation industry has committed to achieve full decarbonization by 2050 under the Green Deal pacts, joined by the European Commission in 2019 (23).

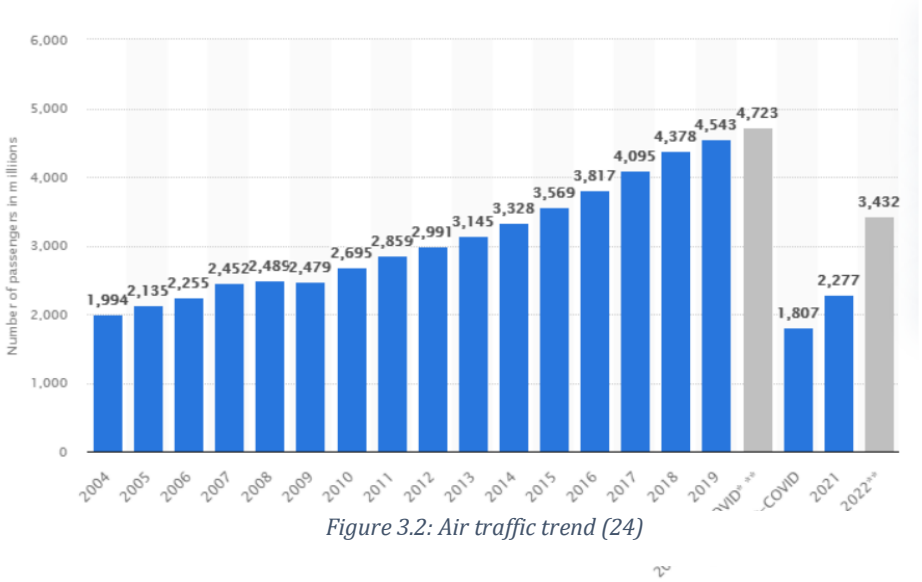

Figure 3.2: Air traffic trend (24)

As a result of this commitment, the FutPrint50 project was initiated to reduce the environmental impact of regional air traffic. The project has 14 international partners based in Europe, and is focused on reducing pollution in regional aviation routes that have seen steady growth since 2003 (23).

The most effective way to reduce harmful emissions in aviation is through the electrification of the power train. However, current technology doesn't allow for the use of electric engines alone, as the specific energy of modern batteries is too low to support aircraft design. The solution proposed by the FutPrint50 project is to partially electrify the power train by combining conventional gas combustion engines with electric engines, resulting in increased efficiency and reduced emissions, as well as a more manageable flight plan (23). Additionally, the conventional engine will use innovative drop-in sustainable fuel instead of conventional jet fuels.

The goal of the FutPrint50 project is to introduce a hybrid-electric regional 50-seats aircraft (HEA) by 2035/2040, which will contribute to the advancement of electric propulsion technology and unconventional aircraft design. This aligns with the European Commission's vision of reducing the environmental impact of the aviation industry.

3.3 Aircraft requirements

For this peculiar project specific requirements have been developed, these requirements are significant in the way that they start putting constraints and indication about the unconventional aircraft design.

What makes possible to identify goals and requirements are the stakeholders and their needs, so the stakeholder analysis and the interaction with them is fundamental to ensure that the project is going the right direction and that the requirements that have been set will eventually validate the overall system.

The stakeholder identified for the FutPrint50 project are (23):

- EU citizen
- Authorities
- Operator
- Airport
- Air traffic management
- Supplier of energy
- Passenger

In this view the HEA can be seen as a system within a system influenced by the surrounding environment

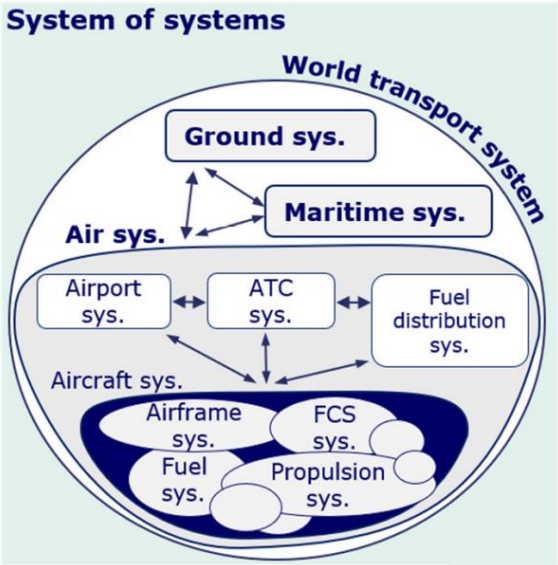

Figure 3.3: System of systems (23)

These top-level aircraft requirements (TLARs) set a first step in the design definition, they are divided in four categories: environment, market, operations, and performance. They are the result of a comparative analysis carried out for this aircraft

In the following tables are reported requirements like they appear in the (23) reference for the FutPrint50 project.

3.3.1 Environmental requirements

TLAR (environment)	Value
Reduction of CO_2 emissions	>= 75% vs ATR42
Reduction of NOx emissions	>= 90% vs ATR42

Reduction of noise emissions	>= 65% vs ATR42
Emissions during ground operations	No emissions
Materials used in the design	Sustainable end-of-life solution
Use of alternative propellants	Yes

Table 3.1: TLARs about environmental aspects (23)

What is desired is not just to reduce emissions during operations but in the whole life cycle of the system.

3.3.2 Market requirements

TLAR (market)	**Value**
Number of passengers	< 50
Cargo capacity	>= 50 kg
Luggage bins	>= 0.06 m^3 / passenger
In-flight entertainment	Seamless connectivity
Cabin altitude	<= 2000 m
Cabin ventilation	>= 0.25 kg/(min fresh air per passenger)
Cabin temperature	23 °C
Cabin humidity	10%
Lavatory	>= 1
Galley	>= 1
Direct operating costs	Competitive with ground transport
Dispatch reliability	>= 98%

Table 3.2: TLARs about market aspects (23)

These requirements have been set to guarantee passenger comfort and to ensure to be competitive on the market compared to other types of transport, also some requirements have to meet international regulation and specifications (CS-25 large airplanes).

3.3.3 Operational requirements

TLAR (operation)	Value
Wingspan	< 36 m
Weather operations	All weather
Turn-around time	<= 25 min

Table 3.3: TLARs about operational aspects (23)

The wingspan shorter than 36 m will allow to be compliant with aerodromes of category 2C, to increase productivity the aircraft will have to be able to operate with all weathers and with a reduced turn-around time.

3.3.4 Performance requirements

TLAR (performance)	Value
Design cruise speed	450-550 km/h (Mach 0.40-0.48)
Design payload	5300 kg
Maximum payload	5800 kg
Design range	400 km + reserve
Maximum range	800 km + reserve
Reserve fuel policy	185 km + 30 min holding
Range from hot and high airports (design payload, international standard atmosphere (ISA) +28, 5400 ft)	450 km + reserve
Range from cold airports (design payload, ISA −30)	450 km + reserve
Take-off field length (maximum take-off mass (MTOM), sea level (SL), ISA, paved)	<= 1000 m
Take-off field length STOL (SL, ISA, paved, > 80% pax)	<= 800 m
Landing field length (SL, ISA, paved)	<= 1000 m
Rate of climb (MTOM, SL, ISA)	>= 1850 ft/min
Rate of Climb at top of climb	>= 1.5 m/s
Time to climb to FL 170	<= 12.7 min
Maximum operating altitude	7620 m

Service ceiling for one engine inoperative (OEI) (or equivalent) (95% MTOM, ISA +10)	4000 m

Table 3.4: TLARs about performance aspects (23)

As said before this aircraft will be competitive for short flight ranges where other means of transport lack infrastructures, for this purpose also the take-off length will be compatible with small airports. The cruise altitude is set to 7620 m for multiple reasons: the turboprop duration mission does not require a higher altitude; with this altitude the cruise segment will be longer allowing a more comfortable travel and a lower fuel consumption due to the reduced climb segment (23).

From the requirements mentioned in section 3.3.1, 3.3.2, 3.3.3, 3.3.4 derive the TMS requirements that will concern this project, work will be done to assess which requirements have a direct impact on the TMS and therefore which are of interest for this project.

3.4 Reference mission

The reference mission stems from the range requirements and more generally from the stakeholders target to be competitive with other ground transports on regional routes.

Since there is the need to test different possible operational conditions, different missions are identified, each one of these will be characterized by a particular demanding flight condition. All these routes take place between real existing airports all over the world.

Flight Mission	**Route**	**Characteristic**
Design range	EDI – DUB	400 km
Maximum range	TJM – NJC	800 km
Cold operations	RKV – EGS	ISA -10
Hot and high operations	MAX – ACA	MEX: 2230 m, ISA +10
Extreme cold operations	YKS – VYI	ISA -40
Mountain operations	LIM – AYP	After take-off: 3000 m at 80 km, 4500 m at 150 km
Island operations	TMS – PCP	Two 200 km-legs without refueling
STOL operations	CVU – HOR	CVU: Runway length 800 m

Table 3.5: Chosen flight missions and their characteristics (23)

This paragraph reports the selected missions because it will be significant in identifying the most critical conditions in which the aircraft would have to work, this is strictly related to the most critical thermal loads that the TMS will have to withstand during its operations.

Here is also reported a typical flight profile for a given mission (Figure 3.4), as it can be seen it includes climb, cruise, descent, approach and at the end it is hypothesised either a go around or, in the image, an alternative flight.

In the following work the flight mission will be discussed in more detail since it will be associated with possible TMS scenarios, for instance in this flight profile initial phases like taxi and engine-start are not considered.

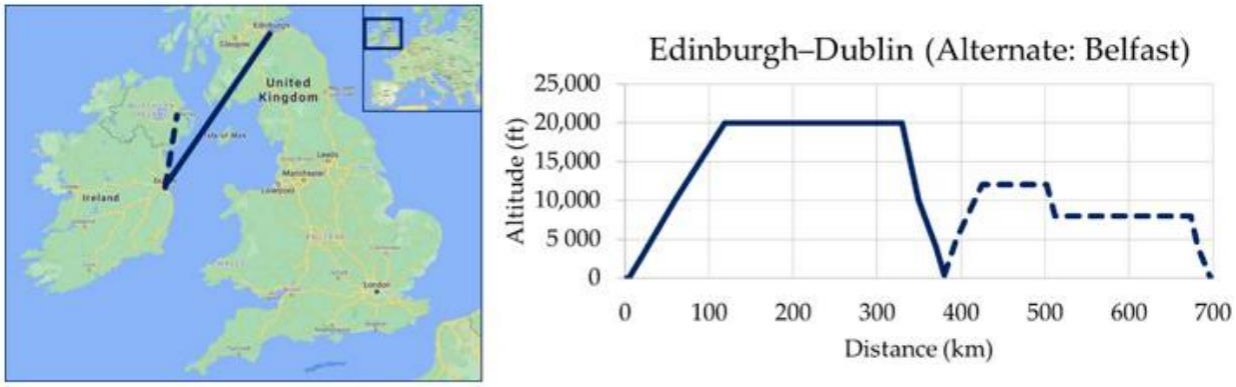

Figure 3.4: Example of flight route and profile of the design range mission EDI–DUB (Edinburgh, United Kingdom to Dublin) (23)

3.5 Aircraft layout

This section presents the aircraft as shown in (27).

From figures 3.5 and 3.6 it can be observed that the aircraft has a high-wing configuration with a conventional fuselage and a T tail design. There are four propellers, the biggest two are positioned closer to the body and the two smallest are at the wing tip, this power train configuration is quite unusual, and it comes with the innovative hybrid concept that this aircraft aims to implement.

The aircraft will have a MTOM of approximately 20 tons.

Figure 3.6 also depicts the most important aircraft dimensions and sizes.

Figure 3.5: Aircraft layout (27)

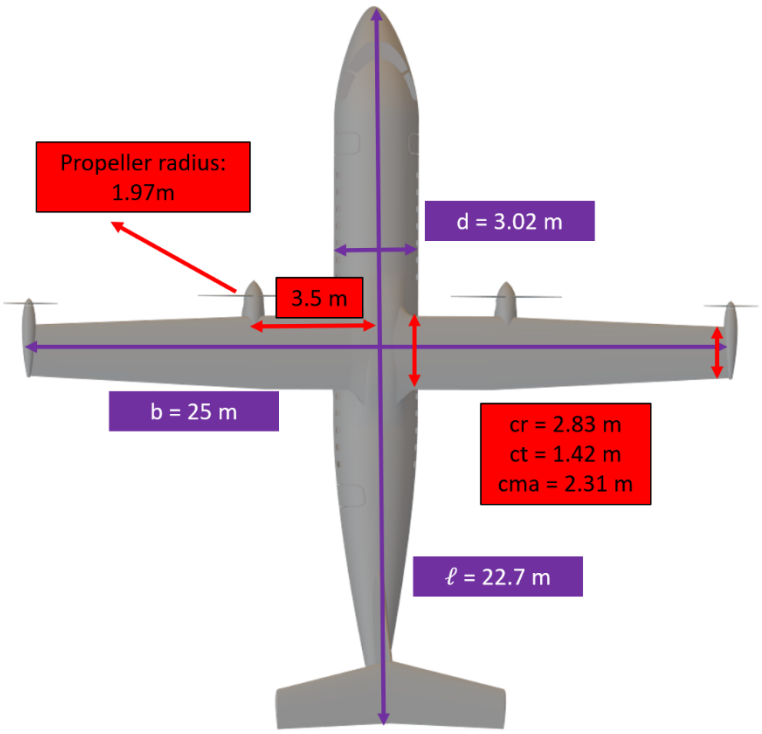

Figure 3.6: Aircraft dimensions (27)

It is expected to board 48 passengers, 2 cabin crew elements and 2 pilots.

As figure 3.7 illustrates the innovative propulsion system is formed by two gas-turbine engines, which drive the two internal propellers, and two electric motors that drive the wing tip propellers. Moreover, the SAF fuel has been selected as a more sustainable way to power the conventional engines. The goal for the FutPrint50 aircraft is to reach a level of hybridization of installed power of approximately 10-30%. The parallel architecture provides also other benefits in terms of redundancy and therefore ensuring safety during operations.

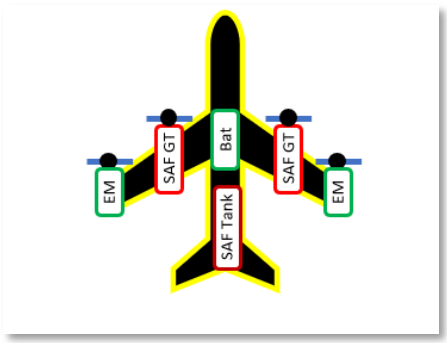

Figure 3.7: Aircraft propulsion layout (27)

The electrical configuration associated to the propulsions system includes motor/generators for all four engines, these components can work in both directions either to generate thrust or to charge batteries in case of need. In this configuration a Power Monitoring and Distribution Unit (PMDU) is needed to direct the energy in the right direction and monitoring the status of all components.

Batteries are also one of the main components of this architecture since they will provide the necessary energy to power electrical engines, the power electronics will drive this energy to the motors which move the propellers.

Batteries can be charged through the generators associated to the turboprops, the alternative solution, in case of need, will be to use the wing tip propellers as turbines to collect energy that would be converted switching the electric motors into generators (figures 3.7 and 3.8). For the batteries the aim is to provide a voltage of 1600 V with a specific energy of 300 Wh/kg, it is expected that the volume occupied from all the batteries will be between a range of 3.7 and 5 m^3.

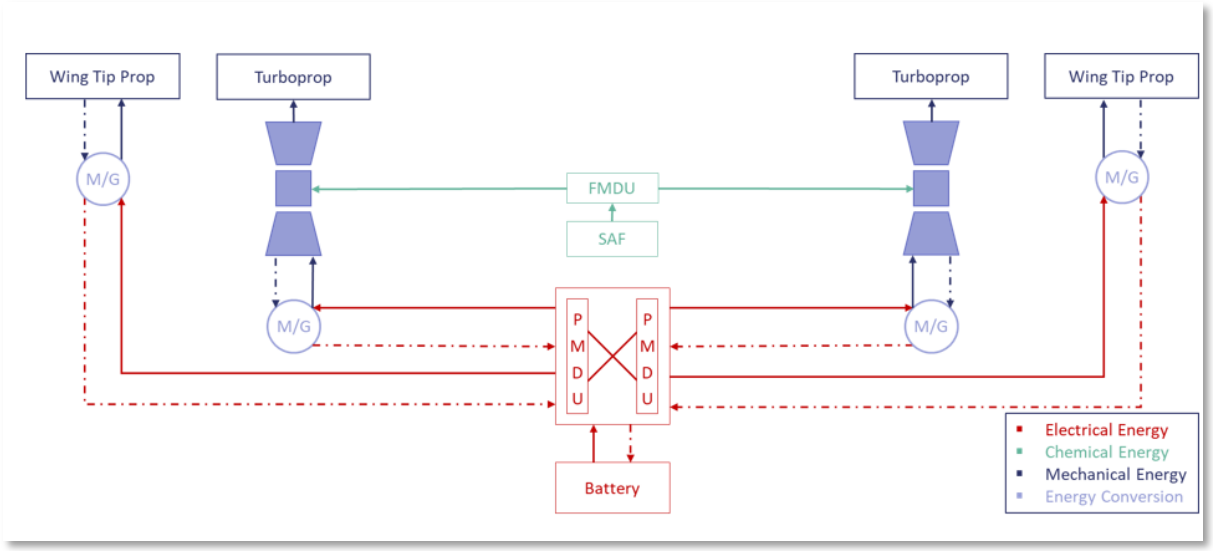

Figure 3.8: Aircraft propulsion layout with energy interconnections (27)

In figure 3.9 is possible to see the main components necessary to the electric architecture. Overall, the installed power of this aircraft will be of 4-5 MW.

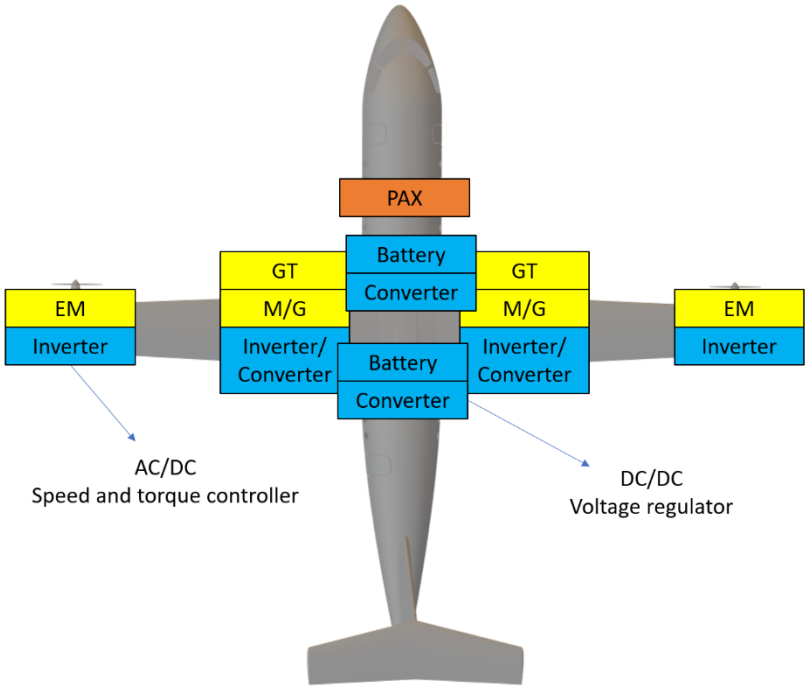

Figure 3.9: Aircraft electric components layout (27)

3.6 Thermal Management System layout

In the reference (27) is also located information regarding the operating temperatures of the components shown in the previous layout, the scheme in figure 3.10 is coherent with the one in figure 3.8, is thus possible to have details about the ideal temperatures demanded to the TMS from different components.

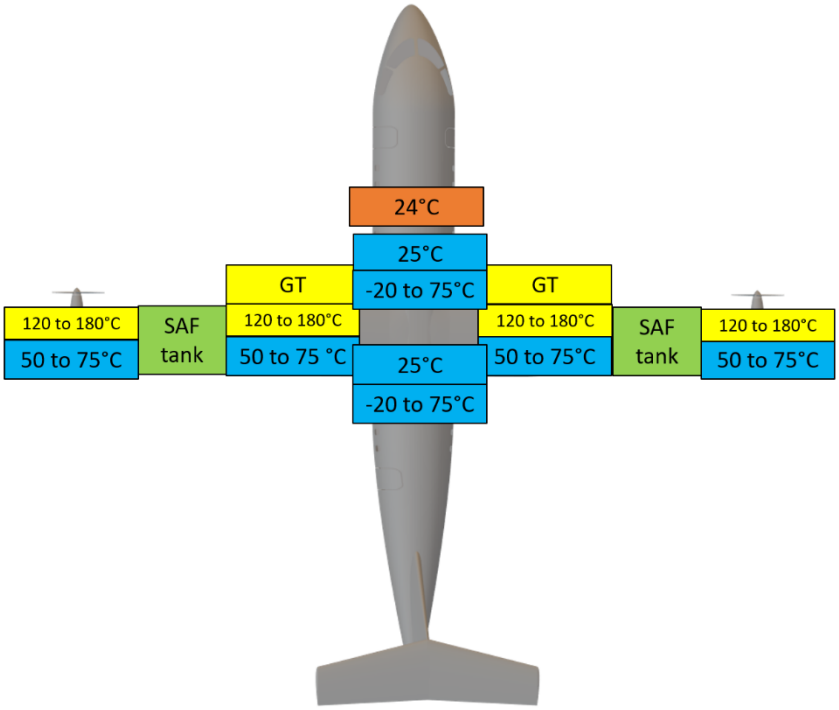

Figure 3.10: Equipment operating temperatures (27)

Element	Temperature range (°)
Electric motor	120<T<180
Motor/Generator	120<T<180
Inverter/Converter	50<T<75
Battery	T≈25
Converter	-20<T<75

Table 3.6: Equipments temperature range (27)

Clearly these are not the only temperatures under which the equipments are able to work, but they are the optimal ones.

The required temperatures data are the base from which the TMS will be developed.

Note that this Thermal Management System regards specifically the components involved in providing thrust through the propeller driven by the electrical motor, so it is a very specific TMS built to cool the “electrical drive train”.

As can be seen from figure 3.11, a possibility is to implement 4 TMS in the configuration, two for the two couples battery/converter in the aircraft body and then one TMS for each wing regulating the temperature of the electric motor, the inverter/converter and motor/generator units.

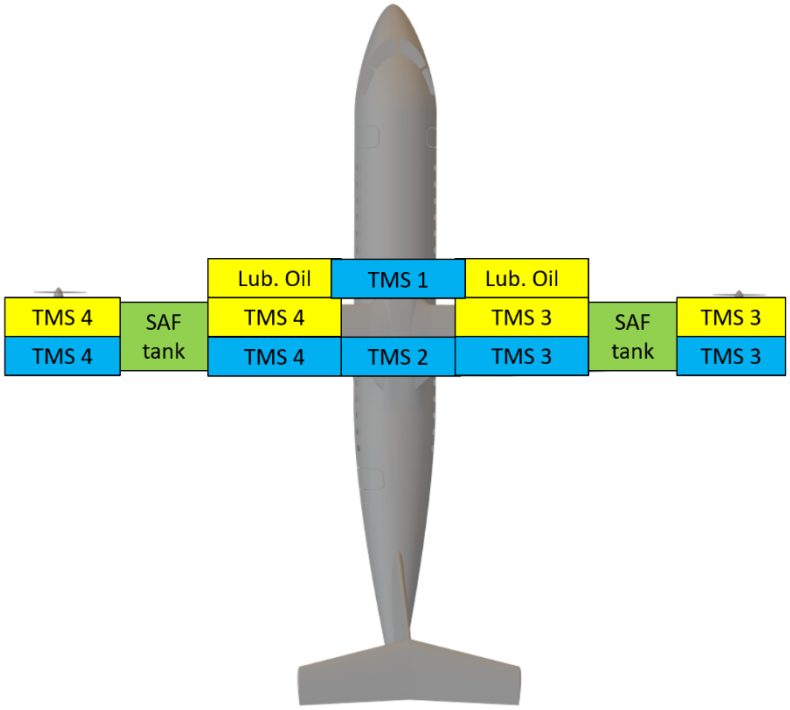

Figure 3.11: TMS layout (27)

In figure 3.12 is depicted an option for the TMS configuration, in this case the equipments, namely inverter and battery, are refrigerated through a Liquid Cooling Cycle (LCC) combined with a Vapour Cooling Cycle (VCS).

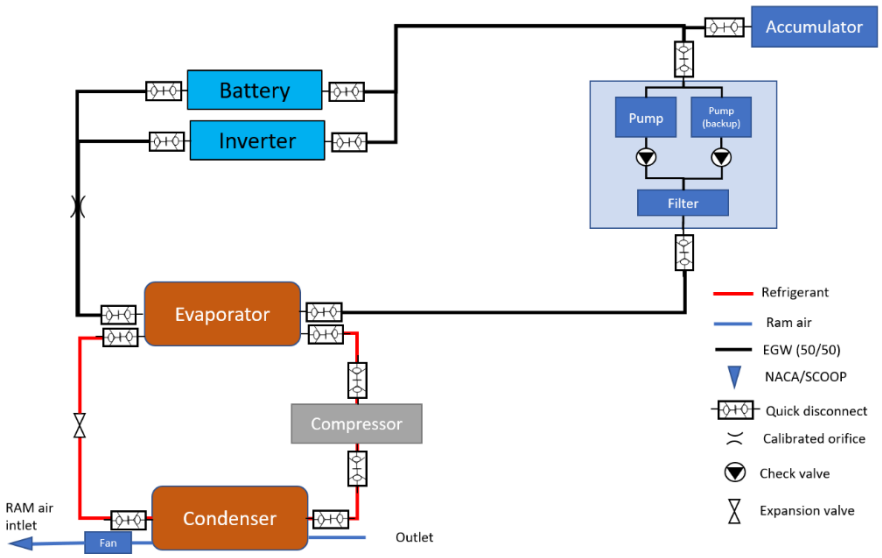

Figure 3.12: TMS architecture option 1 (27)

The LCC consist of an architecture in which the liquid Ethylene-Glycol (EGW) is driven through a pump to the supply line that passes through an evaporator, this element cools down the liquid that then goes to refrigerate the components absorbing the heat loads. In this architecture also an accumulator, a filter and a redundant pump are planned.

The VCS cycle is used to cool the refrigerant liquid, this cycle exploits the properties of different states of the circulating refrigerant. In this case the R314 is driven through a condenser where the ram air cools and condense it (lowers enthalpy), then an expansion valve further lowers its temperature (lowers enthalpy) and then inside the evaporator the R314 absorbs the EGW energy increasing its temperature and evaporating (increase enthalpy), the R314 in aerial state then goes again into the compressor that passes work to the flow increasing again its enthalpy.

The air used in the condenser comes from the external environment through a NACA scoop and a fan.

The architecture also includes different valves that allow a better control of the whole system.

In figure 3.13 is presented another configuration of the TMS, in this case is possible to observe the addition of a Skin Heat Exchanger (SHX) in the liquid cooling line, this element is used to further decrease the liquid temperature boosting the refrigerating capacity of the system.

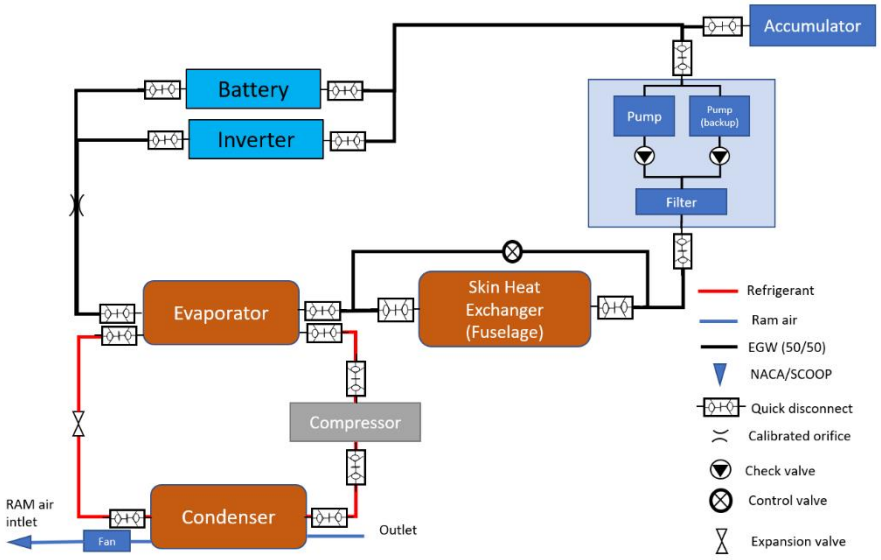

Figure 3.13: TMS architecture option 2 (27)

The skin heat exchanger uses ducts in proximity of the fuselage skin to let the fluid being cooled down from the external atmosphere air temperature.

Other solutions for the TMS design are shown in figures 3.14 and 3.15, in these cases the layout is like the configurations presented above but now the refrigerant liquid passes the heat directly to the external atmosphere through the Ram Air Heat Exchanger (RAHX).

In the latest solution (figure 3.15) is placed again a SHX so that a portion of the heat is removed from the liquid through this component.

In the following sections the text will refer to these options in terms of option 1, option 2, option 3, option 4.

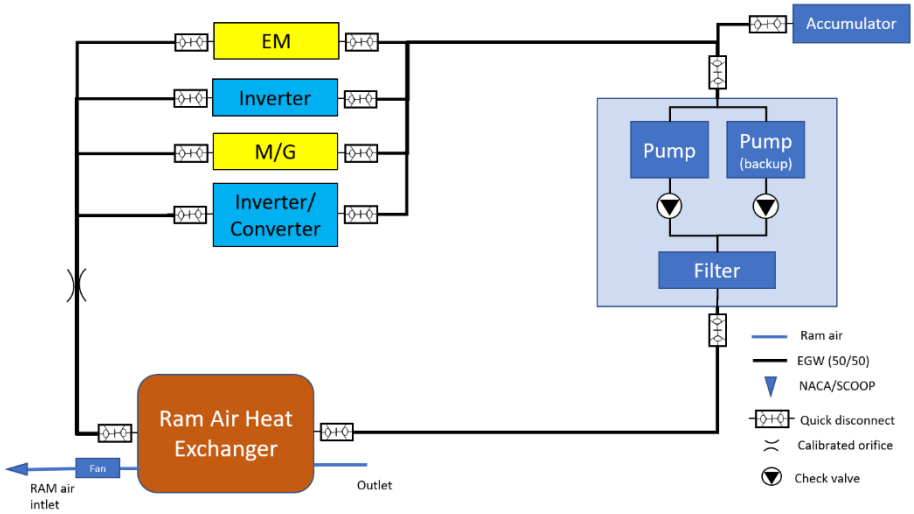

Figure 3.14: TMS architecture option 3 (27)

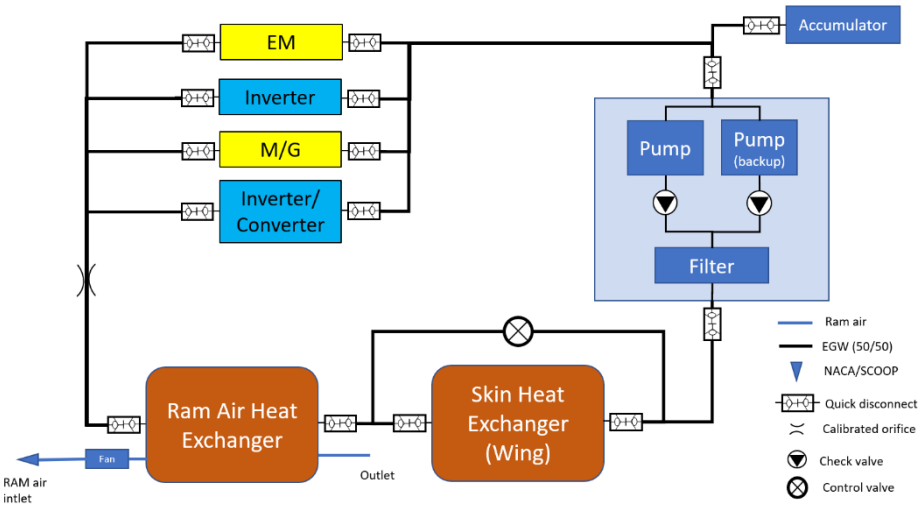

Figure 3.15: TMS architecture option 4 (27)

3.7 TMS requirements

At an early stage of the system development, like the one within which this work is being carried on there are not specific system component requirements yet, these are the ones that directly determine how the components will be built to obtain a successful TMS.

Instead of having component requirements there are some higher-level TMS requirements that derives from a preliminary analysis on the aircraft configuration and on the available technology for such a system. The requirements available and provided by the company are listed in table 3.7.

	Architecture 1	State-of-the-art	Goal 2030
Max thermal power dissipation [kW]	190	N/A	N/A
Specific power dissipation [kW/kg]	0.5	0.16 – 0.26	0.5
Max power required for TMS [kW]	30	95	76
COP [-]	6	1.8 – 1.9	2.5
Max increased drag [cts]	TBD	TBD	TBD

Table 3.7: TMS requirements and targets

As can be seen some parameters still have to be decided and evaluated, in the first column there are the values for a reference architecture, in the second there are values referring to the state-of-the-art technology and in the last column there are the target goals that the system designers aim to achieve by 2030.

The max thermal power dissipation is the maximum heat flow that the TMS shall remove from the subsystems.

The specific power dissipation is a parameter defined as:

$$specific\ power\ dissipation = \frac{thermal\ power\ dissipation}{TMS\ mass} \tag{3.1}$$

This parameter allows to estimate how well the TMS configuration can remove the heat.

The max power required is the maximum electrical power required by the TMS to remove the amount of heat.

The COP stands for coefficient of performance and is defined as follows:

$$COP = \frac{thermal\ power\ dissipation}{TMS\ power\ required} \tag{3.2}$$

It evaluates how much heat the system can manage depending on the electrical power used to work.

The last parameter regards how the TMS inlets and scoops affect the aircraft aerodynamic but it has to be evaluated in the future.

3.8 TMS Matlab/Simulink models

This section reports the models provided by the company and used throughout this project, it shall be considered that during the design process of the system the models are constantly updated, but for the purpose of this work it has been chosen to stick with the version delivered at the beginning of the project.

More precisely Embraer company provided four Matlab/Simulink models, one for each cooling layout option reported in the previous paragraph.

The models carrying out the analysis are built in the Simulink environment, nevertheless every simulation needs to be initialised and thus a Matlab script operates the pre-processing phase

assigning to the model variables those needed values. In particular, the data imported in the Simulink model through Matlab concern:

- Properties of the atmosphere and external air
- Properties of the EGW
- Properties of R314 changing phase fluid (option 1 and 2)
- Flight conditions
- Geometric parameters
- Pressure losses
- VCS parameters (option 1 and 2)
- Efficiencies

The properties of air, EGW and R314 are in turn imported in Matlab from tables provided by the company, is possible to find these tables in Appendix I

In addition, the main parameters which characterize the simulation are the heat load and the target temperature, these values are defined in the Matlab script as well so that the simulation can access them through the workspace.

The Simulink model itself is a static model, it does not perform an analysis of how the system is behaving along the mission, but instead having the data about the mission, the heat load and target temperature, it evaluates in a preliminary way how the TMS affects the aircraft configuration.

The heat load is a value related to heat dissipated from a certain component and therefore that must be removed by the TMS, in the same way the target temperature is a value related to the optimal operating temperature considered for that component. With these numbers set and the description of the system through the parameters imported the Simulink model can evaluate results in terms of:

- Mass
- Power consumption
- Ram air consumption

The results refer directly to the inputs given during the pre-processing, so if the heat load refers to heat that a single component or subsystem produces then the results will refer to a hypothetical TMS which will have to cool just that component.

It is not a goal of this project to perform a deep analysis of the Simulink models provided for this use case, anyway it is worth presenting them and highlighting the most significant features.

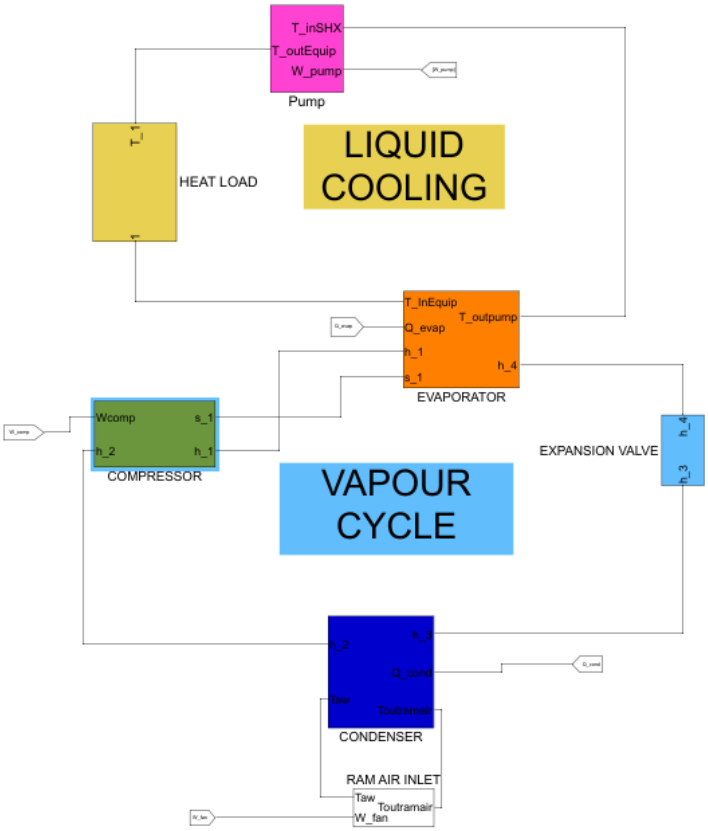

Figure 3.16: TMS Simulink model option 1

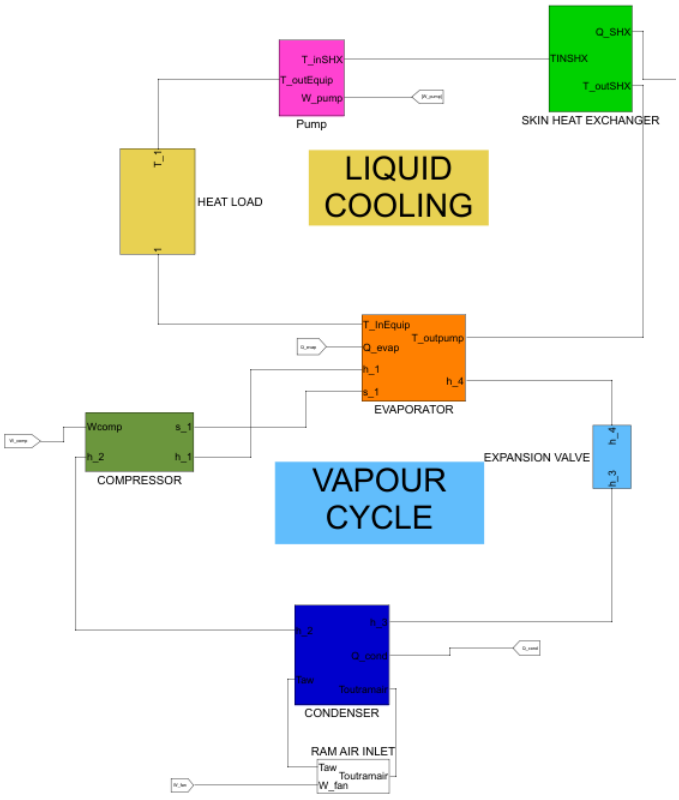

Figure 3.17: TMS Simulink model option 2

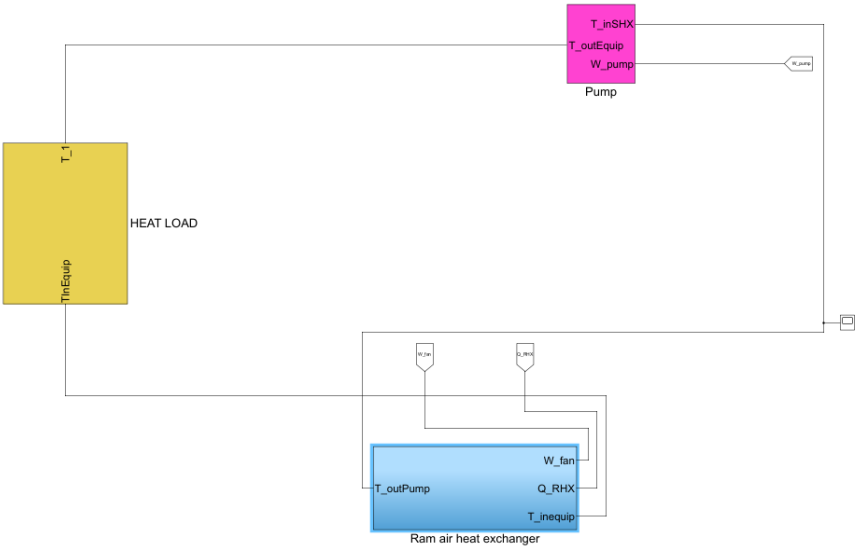

Figure 3.18: TMS Simulink model option 3

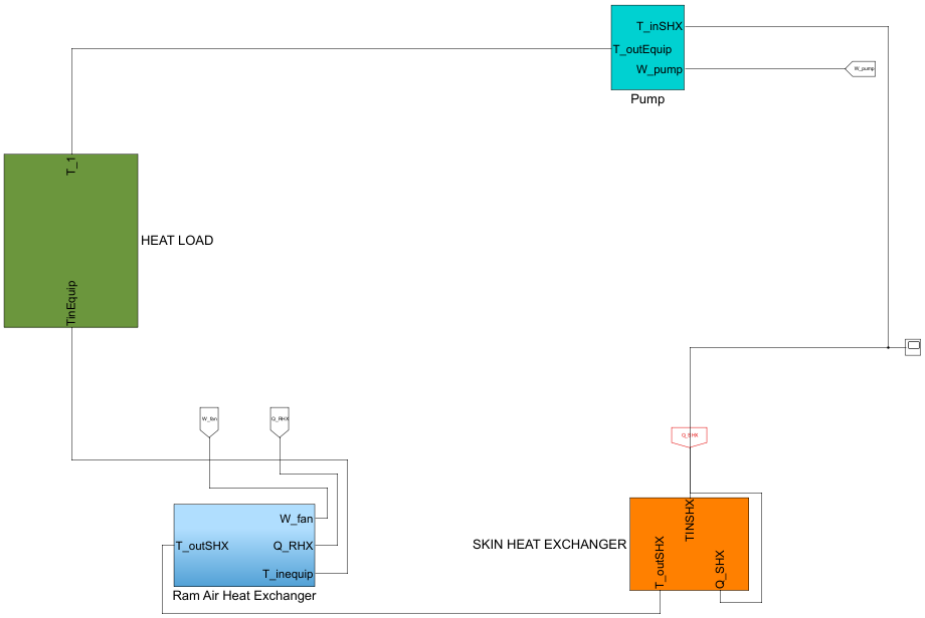

Figure 3.19: TMS Simulink model option 3

These Simulink models were clearly built exploiting similarities with the image representing the layout for each TMS option. It is possible to identify different blocks which works as the related component, although the models do not use simulation capabilities provided by Simulink, many block types and features are used to create sizing models that consider the impact of every component on the TMS preliminary evaluation (lookup tables).

In the models for option 1 and 2 can be seen the two different cycles namely the liquid and the vapour cycle, while for the last two options there is just the liquid cycle. In the same way for option 2 and 4 it can be found the skin heat exchanger component.

It is clear from this design that every key component of the TMS, described in previous sections, is built in the model, and thus simulated.

Anyway, to extract the results from the simulation a post-processing script is provided in the Matlab environment, this retrieves the simulation values and gives the overall results explained before.

The pre-processing and post-processing Matlab scripts are similar for all four cases, in Appendix II can be found an example of these codes.

It must be considered that the aim of this work is not either to modify or discuss the models presented in this section, the project goal is to connect these models to the MBSE environment in a suitable manner.

4 Goals outline

At this stage of the work the research area and the use case are defined, despite that a specific target for the project has not been identified yet. It is known that the main topics are the MBSE model and the connection with Matlab/Simulink, there is now the need to understand in which direction to orient the project development so that the result could be helpful for the engineers working on the FutPrint50 project. We have to clearly understand which capabilities of the MBSE environment to exploit and what we want the model and the connection to do or to verify, another key phase of the work is to understand how the engineers are working on this use case and how the information exchange works between different teams, this to identify what kind of MBSE model they could possibly need.

This section reports the activities carried out on both these sides.

4.1 Research paths

Since MBSE has a wide range of sub-topics is not possible to focus on all of them at the same time, instead it is interesting to concentrate on a specific aspect of the MBSE approach and eventually bring that knowledge to a specific use case such as the TMS considered in this work.

The first part of the project is to research about possible exploration paths that could be followed along the work, the aim in this preliminary phase is basically to gain knowledge about some MBSE aspects and then decide what practises to leverage during the modelling activity.

The research regarded the following MBSE topics:

- Requirements Verification Automation
- Scenario Validation Automation
- Sub-systems Integration

For every path it is desired to know what the “state of art” is currently and if there are either tools or software that could be exploited that are available and what could be free to use.

4.1.1 Requirements verification automation

Requirements verification and validation (V&V) is a fundamental practice in engineering, it must be accomplished effectively to ensure consistent results. There are many tools that allow requirements management, for instance IBM Doors is one of the most known, but not all of them are developed to also support the V&V phase.

Validation and verification basically require that results coming from a set of predefined activities are compared with values and indications written in the first place as requirements. Performing this operation for thousands of requirements may lead to errors and is extremely time-consuming, also considering that the results can come from very different tasks carried out separately from each other. Results can come from experimental tests, inspections, measurements and simulations: in particular, this last source is of interest in this work and at the moment it seems that a seamless way

to link the results of a dynamic simulation and requirements manager tools is not commercially available.

A software supporting V&V graphically help the engineer keeping track of all requirements and results imported in several ways, also changes in both sides are considered and pointed out to the user. An example of this type of tool is the V&V Studio developed by The Reuse Company, but many requirements management tools implement similar features.

The requirement engineer has to derive, starting from high-level customer needs, the specific system and performance requirements. The process to trace every requirement in detail creating a coherent requirement tree is complex and error-prone, furthermore at the end the low-level requirements have to be addressed to specific system components.

To make this task easier and more consistent there is the possibility to write requirements and the whole requirement chain in a simplified and controlled way leveraging Controlled Natural Languages (CNL). These languages consist of a set of rules that drive the user in the writing process, CNLs are exploited by a few requirements management tools, this kind of software provide the user also the possibility to create patterns through which the engineer is forced to keep a coherent form while writing. For example, the RAT tool provided by The Reuse Company enables these functionalities and associated to a MBSE architecture modelling tool like Capella, creates a consistent reference between model elements and requirements (Knowledge Manager), avoiding possible misunderstandings and repetitions while writing them.

All the information reported in this section is brought from sources (2), (3), (4), (5), (6), (7), (8) and university lectures, it is not an aim of this project to discuss them in further detail.

CNL is a key point that could help automating V&V, indeed controlling the language is the first step of translating requirement sentences into values and constraints that could then be addressed to components and managed by an automatic process. As said before, the other feature that seems missing now among requirements V&V tools, is a framework built specifically to link simulation results with requirements and able to analyse the comparison between them.

To conclude this section, it can be assumed that integrating the two points mentioned above within a MBSE tool supporting the development of the model, could guarantee a helpful way to manage the model itself, requirements allocation and traceability, simulation results and requirements verification.

4.1.2 Scenario validation automation

Within the MBSE approach there is the need to describe and model how the system will work in different situations, for this purpose there is the possibility to create various diagrams that represent the sequence of actions that a certain system performs and the states through which the system passes. These types of diagrams are namely Scenario diagrams and Mode and State Machine diagrams.

These diagrams are exploited in MBSE practises to have a preliminary idea of how the system will behave and so having in advance an idea of possible optimization areas and critical conditions.

Once that the project has gone forward there will be then the need to verify if the system is performing like it was assumed in first place, this operation can be carried out either by observing the real system or creating a model that simulate his dynamic and behaviour. This second option is particularly interesting for this project where the desire is eventually a lean and seamless way to validate scenarios through simulations.

Specifically for the Capella tool are being developed solutions that allow to connect the MBSE tool with a simulation environment (Matlab) to validate correctness of scenarios. The solution described in (8) is not commercially available yet but the DESS add-on (9) it is. This tool leverages Mode and State machine diagrams to run test simulations in Simulink environment and creating simulation-based scenario diagrams that at the end could be compared with scenario assumed while modelling.

This solution partially aligns with the intentions of this work and perhaps it could be helpful in the following activity, however it should be noted that this process is not totally automated and for that it must be created a set of rules to decide whether the scenario is validated or not. Those rules could be for instance simulation values that have to be compared with constraints given by requirements, this kind of results could certificate that the system is working as expected and that it is in a certain state. The issue that remains is how to relate constraints to different scenario phases or machine modes and then how to actually make the comparison.

4.1.3 Sub-system integration

Integration of multiple subsystems in a common architecture is a delicate matter, to properly work together different subsystems that interact with each other need to share compatible interfaces, to be placed in a suitable manner and be provided with the means to exchange information or substances.

The work of the integrator engineer is to support the system development from a global point of view, giving indications for the project of the single subsystems that at the end will work together in the same architecture. So, there is the need to check if different parts of the system are being developed to be compatible with each other, at present there is not an actual guideline for system integration, it is known that it can be a discipline itself and that if not considered integration could cause several problems in the system development. The integration is treated with a set of specific requirements precisely concerning those topics of interest for integration, for example interface requirements, dimensions requirements and control requirements.

In (10) and (11) are described activities and suggested operation that the system integrator should perform along all the project development and that would eventually benefit the overall result. These are not rules but just recommended actions that those articles claim to be necessary for an effective integrated system design.

From this project perspective it could be interesting to find a way to early identify those requirements that regard integration to manage them separately if needed and at the end automating their verification similarly to this work purpose for all the others.

4.1.4 Possible work directions

Considering all the topics mentioned in sections 4.1.1, 4.1.2 and 4.1.3 some of the MBSE features that could possibly be helpful for this work are identified, however this first research does not outline completely what will be the direction to take throughout the project, before that there is the need to clarify how the MBSE environment could be used to enhance the engineering activities carried out to design the system. MBSE is a powerful tool but its effectiveness depends on the choice of when to use it to overcome complex design issues and what features to exploit, thus before proceeding the engineering workflow, which occur between the engineers involved in this case, will be analysed, the goal is to identify an area where MBSE can really boost the design process. After that the final decision on the project target will take place.

4.2 Workflow analysis

The purpose of this project is to be useful for the engineers developing the thermal management system, to achieve that goal is therefore important to know how activities are split into different teams. In this section all the people whose work affect directly or indirectly the TMS design are considered, from an operational point of view these teams are stakeholders of the system since they are the very first people to interact with it.

Before speaking about people we have to understand that every team is obviously related to a specific subsystem they are working on, thus it makes sense to first consider the main components which from a functional point of view are connected with the TMS. Furthermore, the use case is specifically focused on the electrical drive train, as mentioned in the third chapter, thus now we are considering a very specific set of components:

- Aircraft:
  This is the global system with which the TMS must interact, it includes most of the following points and its configuration affects all the aspects of the mission and the on-board system. Hence it could be said that basically is the aircraft that with its operational needs gives the high-level requirements and demands from the TMS system.
- Airframe:
  This aircraft component provides the structure to the aircraft and supports all aircraft systems and therefore the TMS; it has to be considered that, from the airframe, depends the lift capacity of the airplane and so designing or modifying it for a certain system is particularly critical. The airframe so imposes volumes, weight constraints, and ensure physical coherence to the TMS. Within the airframe there shall be also all the ducts and elements that will allow the TMS to perform its duty in all the components which need its action.
- Energy storage
  On the aircraft the energy storage is represented by batteries, these shall provide energy for all the sub-systems also when an energy source (generator) is not available. This component so provides energy to the TMS, but it also needs the TMS to regulate its temperature, its operating range was presented in the previous section.
- Electric propulsion system:
  The propulsion system, shown in the previous chapter, has its optimal temperature and hence the electric motors and all the electrical components such as the converters/inverters and all the power electronics will have to be properly cooled. It is also important to consider that from the propulsion system comes the energy that drives the TMS itself, it is therefore

fundamental to maintain the health of these components which harvest the energy from the GT generator and from the external propeller generator in case of need.

- Power transmission:
  Similarly, as the propulsion system, all the electrical components are a key element to drive the energy that powers the TMS, but at the same time the electrical lines will have to be refrigerated.

It is easy to understand now that the main engineering teams that this work is going to consider are:

- Mission team:
  This teams works with the aim to derive from high-level customer needs a possible aircraft configuration, they perform preliminary evaluations analysis and, in the end, giving more detailed information and constraints to the teams developing subsystems.
- Airframe design team:
  Works on the integration of all system in the configuration, this team shall discuss and verify the feasibility of some systems design and eventually gives constraints about some key parameters.
- Energy storage design team:
  Here we are considering the storage of electrical energy, this team evaluates the layout and the technology of those devices suitable for carrying and then providing the electrical power needed from the electrical motors and other components.
- Power transmission design team:
  This team designs the means and the layout so that the electrical energy can flow from one component to another.
- Propulsion system design team
  This teams works to design all those components that transform the electrical power coming from the energy storage segment into thrust, thrust is a key parameter of the configuration and so the entire configuration is sensitive to the outcome of this subsystem.
- TMS design team:
  The TMS was not mentioned among the stakeholders because it is the system of interest itself, anyway this team considers all the cooling needs coming from evaluations of others teams and works to define a suitable TMS configuration.

4.2.1 Information exchange

It is interesting to investigate the data and information that the design teams are exchanging between them, the MBSE model will really be oriented to the kind of information and evaluations that all the teams are carrying out. In this section it is assumed that every team works in a straightforward way starting from some inputs, performing its analysis, and providing the outputs.

To represent the information exchange tools coming from Multi-Domain Optimization Analysis (MDOA) are exploited, the matrix built and reported in figure 4.1 that is a tool that allow to immediately distinguish the inputs and the outputs of a certain block. For our use case the blocks are alle the design teams mentioned above, all the vertical lines connected to the blocks are inputs while the horizontal lines are outputs.

The matrix presented in figure 4.1 follows a waterfall logic: the process starts from the team working on the "outer problem" that is the mission definition to satisfy customer needs, then to achieve those mission goals a certain thrust is required and so on the diagonal the Propulsion System (PS) team works to identify a suitable solution.

Descending on the other subsystems we consider that the electrical propulsion requires electrical power and so Energy Storage (ES) team and then Power Transmission (PT) team are represented in this order. In order to work properly all these subsystems, need to be cooled, and thus the TMS design team works to guarantee the proper operating temperatures for all components.

Descending on the other subsystems we consider that the electrical propulsion requires electrical power and so Energy Storage (ES) team and then Power Transmission (PT) team are represented in this order. To work properly all these subsystems, need to be cooled, and thus the TMS design team works to guarantee the proper operating temperatures for all components.

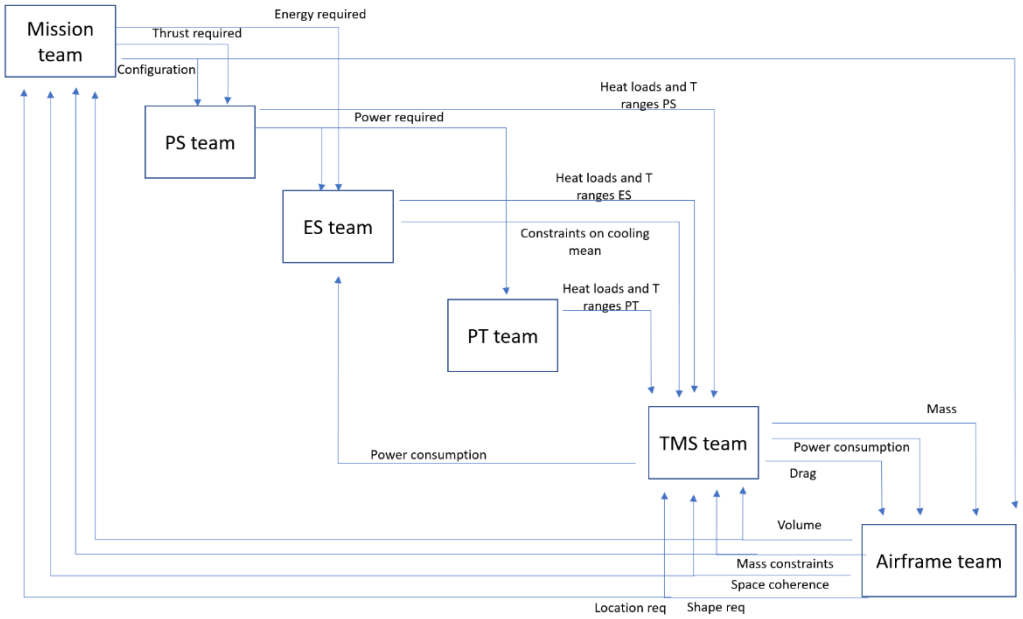

Figure 4.1: Information exchange matrix

At the end the airframe design team works to assess the feasibility of the design and begins a discussion with the TMS and all the other teams to check the integration of all systems and eventually requiring changes in some components or, in the worst case, in the mission.

First the mission design team performs preliminary estimations and gives three outputs of interest for this case: energy required, thrust required and aircraft configuration.

The PS team design the components that produce the thrust required in that configuration and evaluates the electrical power required.

The ES team estimates batteries layout basing on the power and energy requirements.

The PT team similarly estimates all the components that transfer electrical power.

PS, ES and PT teams all provide to the TMS design team the heat loads that must be absorbed and the optimal operating temperature range for each of their subsystems. The TMS design team basing

on this information and considering some specific cooling constraints (for instance on the cooling mean for the batteries), uses the Matlab and Simulink static model to evaluate TMS mass, ram air consumption and power consumption.

The airframe team gets the TMS team results and works to estimate the integration of the TMS into the configuration, also they start discussing with the TMS team about volume and mass constraints, space coherence, shape and location requirements.

The TMS also slightly affects the batteries layout with the power that it requires so the ES team must consider it.

The airframe team at the end gives back their outputs to the mission team so that evaluation on mission feasibility could be made.

The matrix in figure 4.1 is “mission centred” which means that from the mission comes before every other analysis, in another way it is interesting to build also a “TMS centred” version of the matrix, this is depicted in figure 4.2.

In this matrix is possible to clearly identify all the variables that the TMS design team uses as input and how the TMS results affect the other teams work in a waterfall logic also used before.

Thus, the MBSE model will have to enhance this information exchange boosting the TMS development, the key parameters are the ones closely related to the TMS design team.

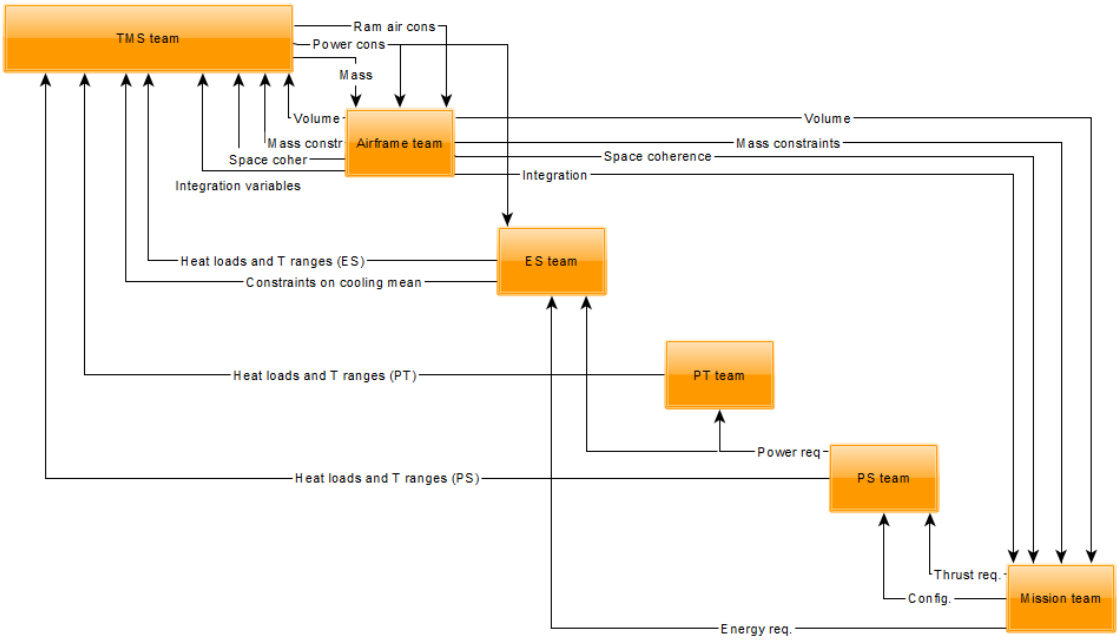

Figure 4.2: "TMS centred" information exchange matrix

4.2.2 Work flowchart

The key information that we want to know to properly direct this work is understanding at what stage of the system development the TMS MBSE model will be placed and used. The TMS modelling

activity would significantly change depending on how deeply defined the system architecture is at this stage of the work.

To better highlight the TMS project workflow a flowchart of all the activities is built and depicted in figure 4.3.

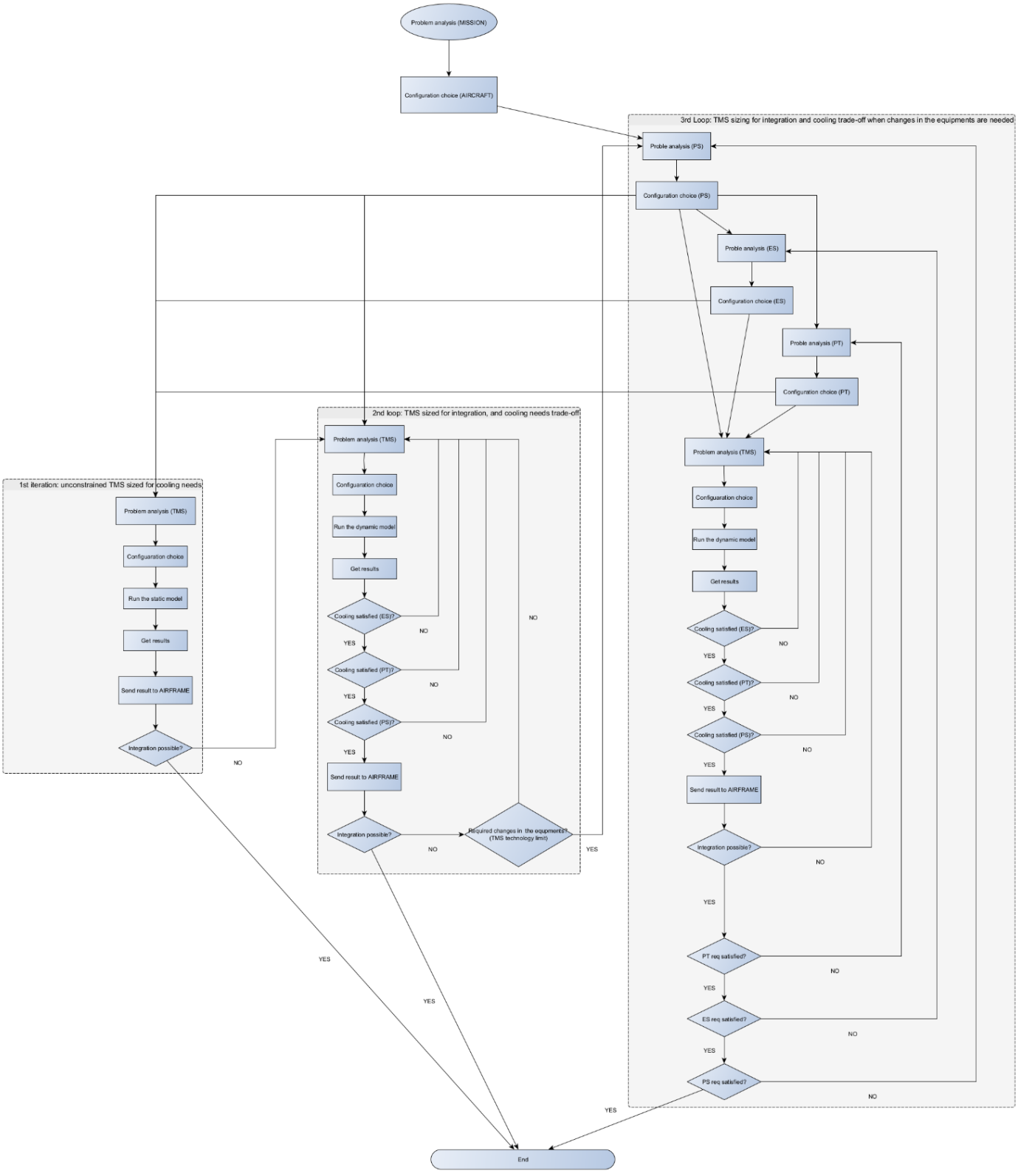

Figure 4.3: Work flowchart

The starting point of all the activities is obviously the mission analysis, then the activity flow follows the path explained in the previous paragraph, what it is interesting to note is that after the configuration choice for PS, PT and ES there is a first "path" followed by the activities.

This first loop is "unconstrained", this means that no trade off analysis is performed, in this phase the TMS design team gets the cooling needs from the other subsystem requirement and gives a solution that can reach that cooling capacity, the estimations about this hypothetically unconstrained TMS are done through the Matlab/Simulink static model. At the end of this first iteration the TMS configuration is verified against integration requirements with the airframe design team, if integration is feasible the process goes directly to the end since the system will be built in this way.

Realistically this is not what happens in real life, the first iteration scope is to see if the work and the TMS layout is being developed in the right direction, usually then it begins the second loop in which trade-offs between integration and cooling needs are performed to obtain a TMS that could be fitted in the aircraft and that cools enough all the equipment.

This second loop could end if a suitable solution is found within the technology limit of the system, this means that if a solution cannot be found changing either TMS layout or components or materials then the process cannot be closed, and a third iterative loop is required.

This third loop also involves the choice of PS, PT and ES configurations, indeed if the second loop do not solve the situation there is the need to change something in the other subsystem configurations until we find that a certain equipment configuration can be cooled by a certain TMS layout.

Note that in the second and third loop the cooling requirements and the temperature ranges are not the only inputs, so the first thing to verify in these cases is that the cooling capability of the layout. To verify the cooling requirements there will be the need of a dynamic model that once simulated could predict the behaviour of the TMS throughout the mission.

Hypothetically there is also another loop not considered in the flowchart, indeed the worst case is that not even changing all subsystems configuration a suitable solution can be found, in this situation there will obviously be the need to reconsider the high-level requirements and the feasibility of the mission with that aircraft configuration.

From this analysis is clear that this project is at its early stage and furthermore a dynamic model of the system has not been built yet, this means that this project is collocated at the beginning and thus in the first unconstrained iteration of the development.

A feasible goal is thus to support the TMS design team and all the other teams interacting with them by creating a suitable MBSE model that will allow faster information exchange and will make easier to check and share the model results.

4.3 Target definition

The goal of this work is to support the design of a thermal management system through the creation of a sort of tool that allows the seamless interconnection between the digital system model and the Simulink simulation model. This project aims to facilitate the tasks of system designers and model developers accelerating the requirements verification process, the target is to create the code that

links MBSE with the simulation environment. This tool will be tailored on the TMS, and the project will also regard the modelling activity and the identification of suitable scenarios to simulate where then requirements will be checked.

This work is collocated within those research projects whose purpose is to gradually introduce the MBSE practise in the engineering environment and aims to facilitate this transition showing all the possibilities that this new paradigm brings.

This section also illustrates the goals of the project and defines which are the steps that have to be taken to get to the result.

As said before the TMS engineers needs that give the motivation of this project are:

- A digital and agile model that could be shared within design teams.
- A clear relation between requirements and the model.
- To identify suitable scenarios where to implement requirements verification.
- Connect the MBSE and Simulink models in a seamless way to allow scenarios validation.
- A code that implements a preliminary requirements verification tailored on the TMS.

In this way the MBSE environment and more specifically the Capella software will be exploited to create a "tool" that will allow a faster communication between all the engineers working on the project.

This tool will be formed by the formal system specific architecture built in Capella and by the connection with Matlab/Simulink models.

First a coherent architectural model of the aircraft will be built in Capella, then sticking on an aircraft perspective, the architectural model will be focused more on the "TMS for electrical drive train" subsystem.

With a solid architectural model there will be the possibility to place and allocate the requirements and constraints into it. At this point the MBSE model will be linked with the Matlab/Simulink model, the connection will be built through Python programming language so that P4C capabilities could be exploited.

A peculiar aspect of the modelling part of the project is that it will be performed in parallel with the system development, usually MBSE helps during the conceptual design phases of the system architecture definition, but now hypothesis on the configuration are available and were presented in section 3. Thus, in this case the design activity has already started and therefore the conceptual MBSE model will not precede the other engineering steps, moreover we shall consider that the Matlab and Simulink models are already available and thus the MBSE model will need to consider what kind of system simulation will relate to.

To manage both the model from an aircraft perspective and keeping it coherent with what is being developed by the engineers, a sort of "bottom-up" approach will be fundamental for achieving a successful result.

In the end it will be interesting to perform a preliminary requirement verification against the parameters set as requirements and allocated in the model.

Hopefully this project and this tool will be useful for the FutPrint50 engineers and will connect in a seamless way the MBSE world with the MBD environment (Matlab) so that all the engineering activities, from high-level mission definition at an operational level to the low-level system simulation, are connected.

Another motivation for this project is that this approach could be used in the future for other aircraft subsystems, now the work is focused on the TMS use case but in the future other tools tailored on other subsystems could be built. This will be allowed by building the model from an aircraft perspective that will create a sort of common base layer from where then different works could focus on different subsystems.

Since the focus will stay on the conceptual design of the system some assumptions will be made throughout the project, it will be important to point them out highlighting how they affect the system development, what analysis they allow to perform and how at the end they impact on the component design in future work stages.

5 Building the digital tool – Phase one

In this section all the activities that lead to the construction of the digital MBSE tool that will support the engineering work will be described. The actions foreseen to get to a consistent result were listed before.

The process starts from building the model for the aircraft, this methodology has been chosen to keep the aircraft perspective all along the project, the aim is not just to model the TMS but to place it the aircraft formal framework.

5.1 Capella aircraft model

In this first part the aim is to build a Capella model from an aircraft point of view, this process will follow the Arcadia methodology with all its steps. We have to consider that the level of detail of the model is strictly related to its purpose, so when the model will be detailed enough to place all requirements and values the process will stop.

This model is built considering the aircraft subsystem division explained in (29), to perform the operational analysis, the systema analysis and the logical architecture there is the need to find a solid reference that describe the aircraft from a functional point of view, this is found in (28).

In (28) aircraft activities at all levels are represented and grouped in functions and subfunctions, these will be the ones used in the model and placed in the respective aircraft subsystems.

5.1.1 Operational analysis

To perform the operational analysis, it must first be considered what is the purpose of the system that is going to be built, since the operational analysis is a process focused on what the system needs to achieve we assume for this very first that the aircraft is not identified yet as a possible solution for the problem.

The "problem" that drives the need of a system is to perform a fast and effective transport on regional routes, this was expressed also in section 3, so from an operational capability point of view there is just the need to move passengers on those ranges.

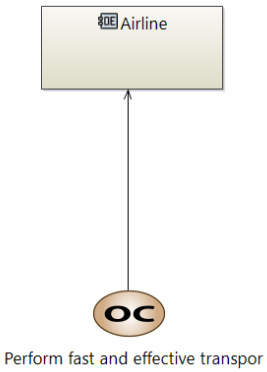

Figure 5.1: Aircraft operational capability diagram

At this level it is assumed that the main stakeholder who seeks to solve the problem is the airline, indeed the need of such a transport directly comes from market analysis that the airlines carry out to provide always better transport solutions.

The airline is the transport provider, the main actor who wants to solve the problem of a fast regional transport on difficult routes and aims to earn money to be economically profitable.

Obviously, the solution for this kind of transport starts to be oriented to a sort of "flighting device" capable of transporting people and things.

In (28) there is a division in some very high-level functions for all aircraft activities, for this case there will be considered all the functions grouped into the "perform air transport mission" function, thus excluding functions like manufacturing, facilities, and support.

These functions are the ones depicted in the operational activity diagram (figure 5.2), as it can be observed they are very general and at this stage they could be easily addressable to another type of flight transport rather than an aircraft.

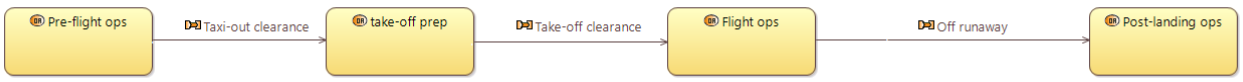

Figure 5.2: Aircraft operational activity diagram

Again, for the purpose of this work we are not considering the functions that at this level regard specifically the mission and the situational scopes.

In the Capella environment at operational level there are no actually functions but activities, anyway the concept of high-level function could be directly translated into a Capella activity.

The sequence described in figure 5.2 regards essentially all the phases that every flying system must perform. It involves all the nominal activities after the manufacturing phase and before the removal from service.

- Pre-flight operations:
  This activity involves all the operations carried out on-ground to prepare the machine for the flight.
- Take-off preparations:
  This activity sequence begins when the vehicle is pushed away from the gate and ends when the take-off clearance is given, this involves all the activities performed during the taxi phase.
- Flight operations:
  It is the main flight activity of the flying vehicle, and it involves all flight phases from take-off to exit runaway.
- Post-landing operations:
  This activity involves all sub-activities from the moment when the flying transport leaves the runaway to when the vehicle is stationary, this includes similar sub-activities and functions to "take-off preparations".

Capella gives us also the possibility to create a diagram where can placed all the stakeholders that will interact with the system that we want to build. This diagram is called entity diagram and is represented in figure 5.3.

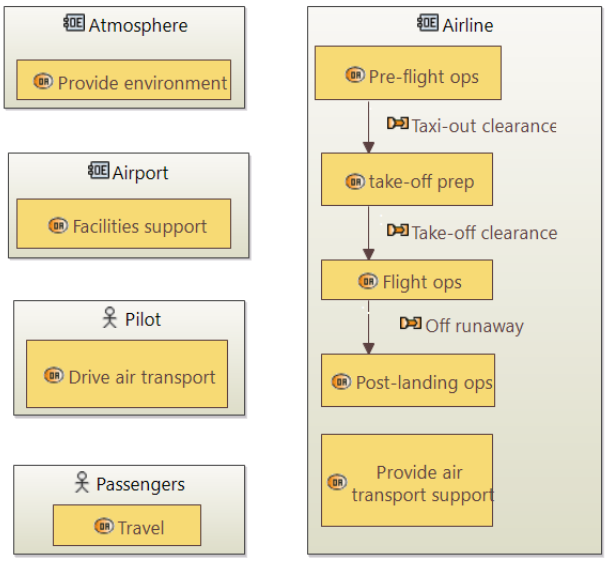

Figure 5.3: Aircraft operational entities diagram

For this operational analysis there were made a lot of assumptions and one of them is that the system will have connections with just the entities present in figure 5.3.

Obviously, the Airline at this stage is the main entity involved in all the operations and activities, it provides the transport mean, namely the flying vehicle, and manages the air transport mission.

The other entities considered in this Capella layer are:

- Atmosphere, that forms the environment through which the mission will take place.
- Airport, that is the starting and arriving facility for the vehicle.
- Pilot, who drives and manages the flight.
- Passengers, that are the payload that need a transport mean.

With the available information these were the only diagrams created for the Capella operational layer, going down through the other layers this project and system will be described in further detail, this kind of process follows the ARCADIA methodology where every Capella layer has its own level of detail.

5.1.2 System analysis

In the system analysis the scope is to identify what the system of interest will have to achieve to satisfy the needs expressed in the operational level, it is at this stage that ideally, we start proposing a “solution” that will solve the regional air transport problem.

Speaking about capabilities we start detailing a bit more than what we had at operational level, as said before the mission of this project is to “perform fast and effective transport”, but then every

system actor has its own commitment. The pilot must drive the vehicle that will have to allow passengers transport, the airline is the main actor that provides this transport service.

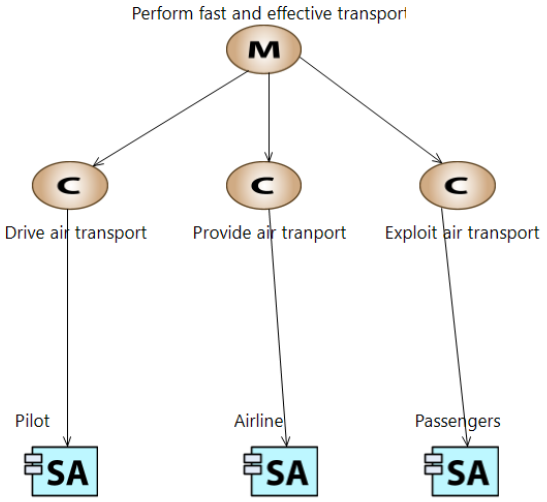

Figure 5.4: Aircraft system mission and capability diagram

Thanks to (28) it is possible now to explicit those activities considered at operational level, as it can be seen from figure 5.5, we are making explicit functions for "take-off preparation" and "flight operations", this because "pre-flight operations" are out of the scope of this project and on the other hand "post-landing operations" are like "take-off preparations".

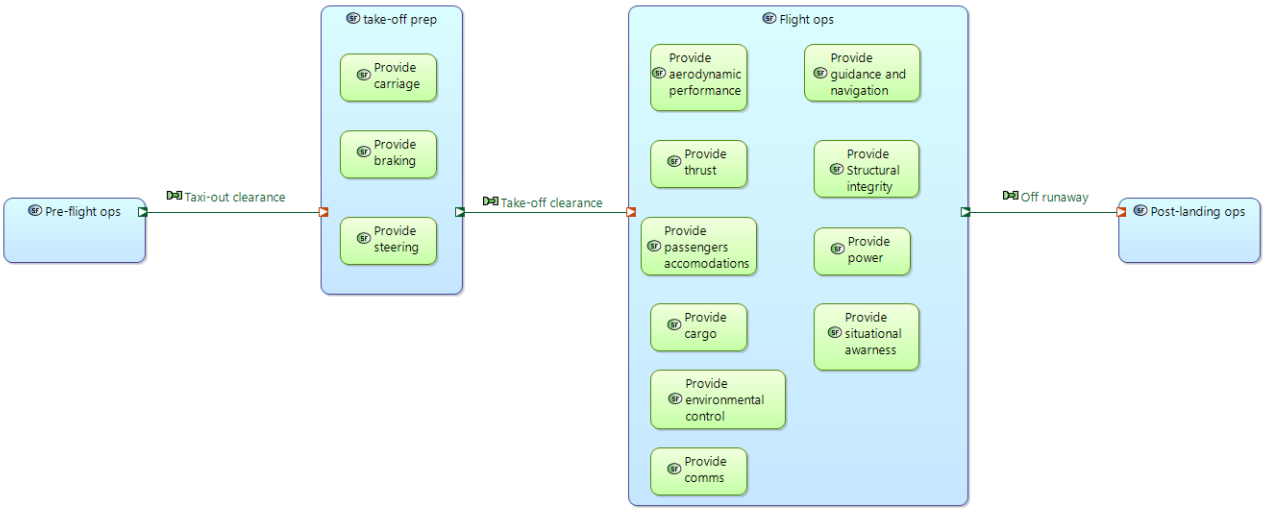

Figure 5.5: Aircraft system functional dataflow diagram

Take-off preparations:

- Provide carriage: the system must provide a mean to move on the ground.
- Provide braking: the system must provide on-ground brake capabilities.
- Provide steering: the system must provide on-ground steering capabilities.

Flight operations are the core functions for the flying vehicle, in this step we are not proposing yet a system, but we are specifying in further detail what such a system shall do during this kind of operations.

Flight operations:

- Provide aerodynamic performance: the system aerodynamic shall allow to fly.
- Provide thrust: the system shall have propulsion means.
- Provide passengers and crew accommodations: the system shall have suitable accommodations for people on-board.
- Provide cargo capability: the system shall provide the space where to place payload.
- Provide environmental control: the system shall maintain an environment suitable for people.
- Provide communications: the system shall allow incoming and outgoing communications.
- Provide guidance ad navigations: the system shall provide guidance and navigations instruments.
- Maintain structural integrity: the system shall resist external loads without affecting its integrity.
- Provide power: the system shall provide all kinds of power to all the subsystems that need it.
- Provide situational awareness: the system shall have the possibility to analyse the situation around itself.

All these functions, listed also in (28), could refer to a very "general" flying vehicle, but it is at this stage that we start proposing a system that could include all the functions mentioned above, for our case this is the unconventional FutPrint50 aircraft.

In figure 5.6 the Capella system architecture diagram gives the possibility to represent the system of interest, namely the aircraft, and all the other actors involved in its operations.

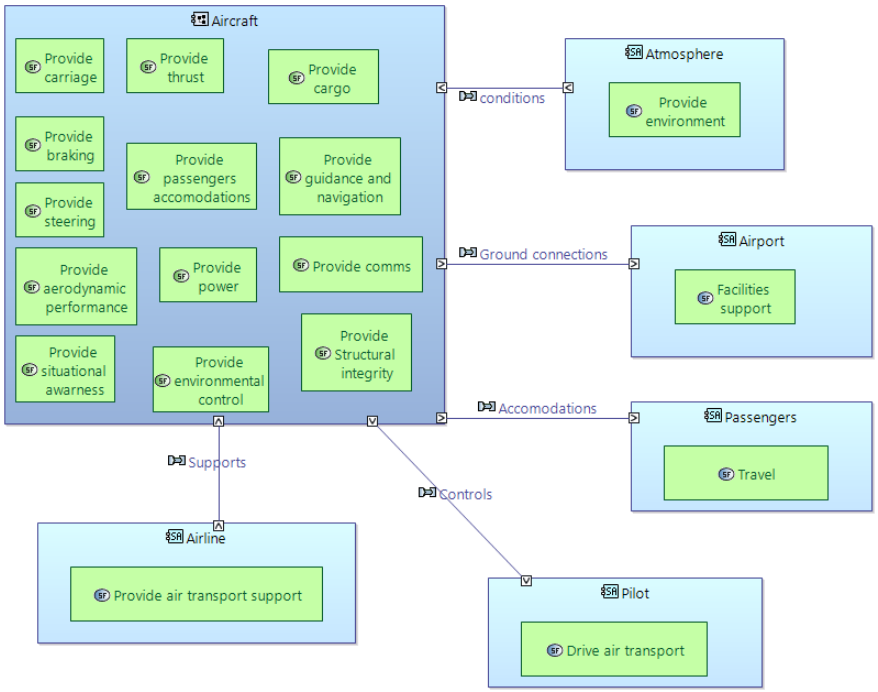

Figure 5.6: Aircraft system architecture diagram

In this type of diagram is then possible to allocate all the functions created at system level, now the airline is still the transport provider but it supports the aircraft that is the system performing the air transport. The connections with all the other actors are made coherently with what discussed before: the atmosphere is the environment around the aircraft, the airport provide the facilities to allow the ground connection, the pilot has his controls to manoeuvre the aircraft, the passengers are seated in the aircraft accommodations.

5.1.3 Logical architecture

In the logical stage of the process, we try to define how the system will be built to satisfy what expressed in previous layers. First, we use (28) to make it even more explicit the functional division, we consider just the "flight operations" phase as it is the most representative for this project.

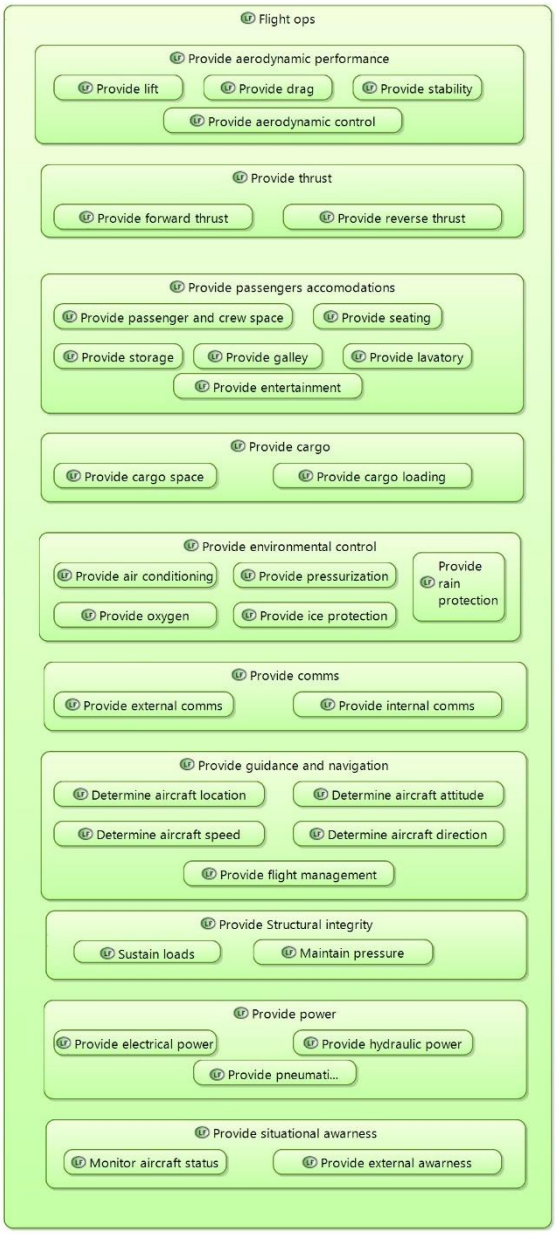

Figure 5.7: Aircraft logical functional dataflow diagram for flight operations

On this Capella layer we see more detailed subfunctions that carry a very specific meaning, each of these requires that the system architecture provides subsystems or components to fulfil system needs.

The functions at logical architecture level are:

- Provide aerodynamic performance:
    - Provide lift.
    - Provide drag.
    - Provide stability.
    - Provide aerodynamic control.
- Provide thrust:
    - Provide forward thrust.
    - Provide reverse thrust.
- Provide passengers accommodations:
    - Provide passengers and crew space.
    - Provide seatings.
    - Provide storage.
    - Provide gally.
    - Provide lavatory.
    - Provide entertainment.
- Provide cargo:
    - Provide cargo space:
    - Provide cargo loading.
- Provide environmental control:
    - Provide air conditioning.
    - Provide pressurization.
    - Provide oxygen.
    - Provide rain protection.
    - Provide ice protection.
- Provide communications:
    - Provide internal communications.
    - Provide external communications.
- Provide guidance and navigation:
    - Determine aircraft location.
    - Determine aircraft attitude.
    - Determine aircraft speed.
    - Determine aircraft direction.
    - Provide flight management.
- Provide structural integrity:
    - Sustain loads.
    - Maintain pressure.
- Provide power:
    - Provide electrical power.
    - Provide hydraulic power.
    - Provide pneumatic power.

Provide situational awareness:

- Monitor aircraft status.
- Provide external awareness.

Changing the focus on how we need to build the system we start evaluating which subsystem the aircraft will need, for this purpose is possible to find in literature information about main aircraft subsystems (29).

The main subsystems are:

- Airframe
- Landing gear
- Electrical power system (EPS)
- Hydraulic system
- Pneumatic system
- Propulsive system
- Environmental control system (ECS)
- Ice protection system (IPS)
- Flight control system (FCS)
- Communication system
- Navigation system
- Surveillance and identification system
- Vehicle management system
- Fuel system

It comes then natural to match the aircraft functions with the aircraft subsystems, addressing the functional need to its own subsystem that will provide that system behaviour. This is shown in figure 5.8.

A difference from what was retrieved from (28) is that the "store fuel" functions was added to give to the fuel system its proper scope.

Obviously these are just some of all the functions that every system performs during aircraft operations, but it is still interesting to note an almost perfect match between functions and subsystems although the references do not belong to same papers. This works in this way because now we are considering just the fundamental functions for every aircraft system.

It can be observed that in this section it has not been mentioned the thermal management system yet, in the following work the model focus will slide to the TMS, but the framework will remain the one developed for the general aircraft system.

The importance of creating the aircraft model stands into the approach the functional analysis of the TMS shall be coherent with what brought from (28) and (29). If here functions and subsystems were brought from literature, from now on developing the TMS model will require the creation of new elements and functions, but these will be placed in the same Capella aircraft model and will relate to some of the subsystems here created.

At this point of the ARCADIA methodology, we realise that is the moment for the thermal management system to appear inside the aircraft model, therefore the work will stick on the logical

architecture layer but changing the focus on the TMS. There is not the need to go into the physical architecture and EPBS levels since no other useful information will be added.

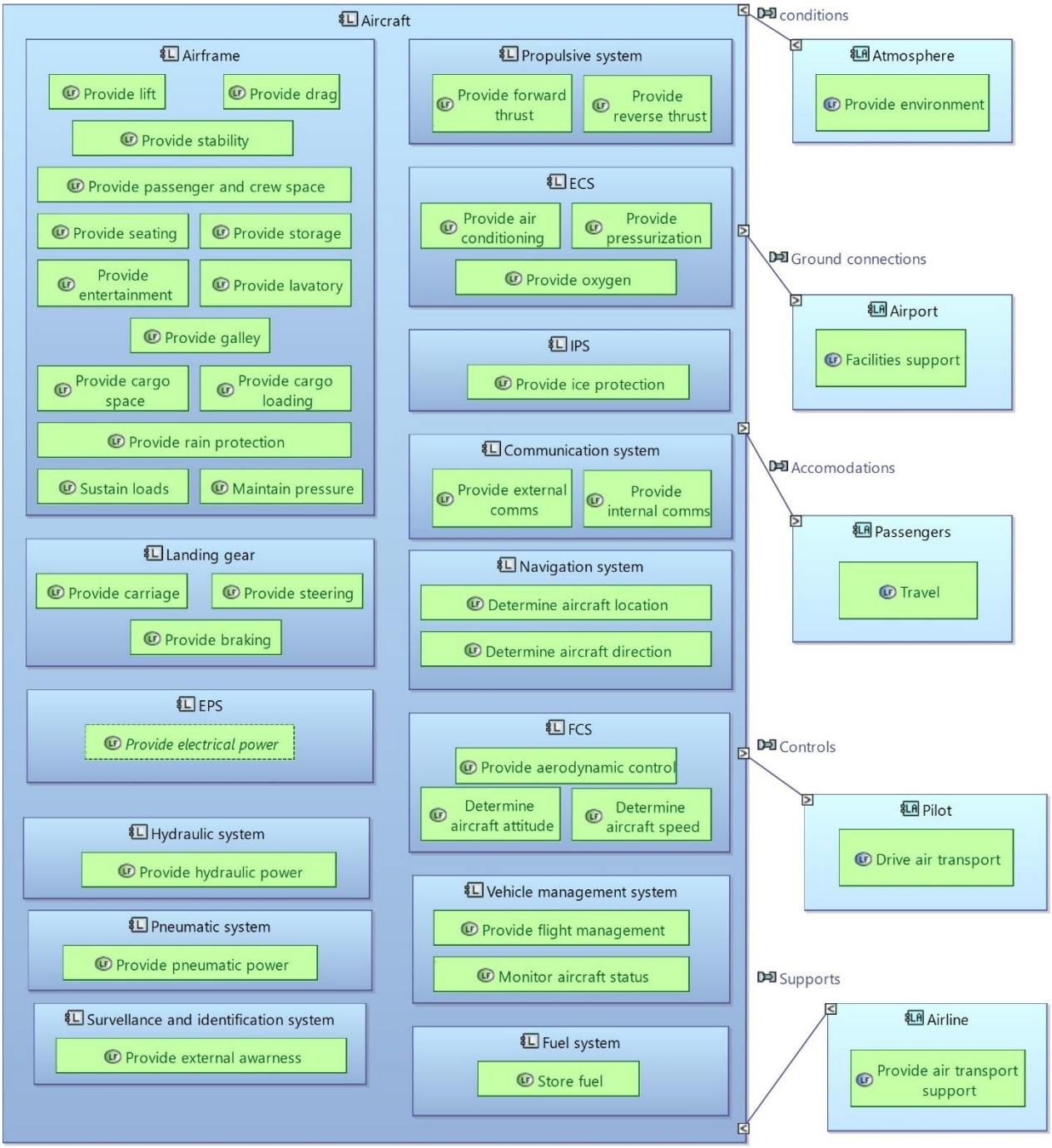

Figure 5.8: Aircraft logical architecture diagram

5.2 Capella TMS model

In the aircraft model presented before the thermal management system does not appear, for our use case although the thermal management system is needed to cool down the electrical drive train. This part of the work consists in analysing the TMS information (section 3) and translating it into a

Capella coherent model. Also, this activity is a sort of bottom-up engineering process because on the other side the sizing models are already available in Matlab/Simulink and thus the TMS architecture in Capella will have to consider how the system is being evaluated exploiting also what is possible to deduct from the numerical models.

Since the goal is to place the TMS inside the aircraft architecture there is no need to repeat all the ARCADIA steps also for our system of interest These reasonings bring to the conclusion that the aircraft logical architecture will be modified in order to consider specially TMS activities and interactions with other subsystems, thus the TMS modelling work will start from the logical architecture level.

It is important to note that this approach can be reproduced also for all other aircraft subsystems.

The first thing to consider is that the TMS will be an aircraft system and so it will be represented as a subsystem block (logical element). Since the terminology TMS is very "general" the assumption is to work just with the TMS for electrical drive train.

During the discussions with the TMS engineers it emerged that ideally the TMS will group under its name subsystems like the IPS, ECS and in the end TMS for electrical drive train, this last subsystem is the one that this project focuses on.

It is now time to consider, in the aircraft architecture, just those elements that have a direct interaction with the TMS, coherently with what presented in section 3 and 4 the main subsystems that are connected with the thermal management for electrical drive train are:

- Propulsive system
- Electrical energy storage
- Electrical power transmission

While the propulsive system was already created building the aircraft architecture, the last two elements are not in the model yet. However, is possible to observe that electrical energy storage and power transmission could both be place inside the EPS element as they are EPS subsystems.

From a management point of view it makes sense also to consider the vehicle management element because in the future there will certainly be the need to implement a TMS control strategy.

The only external actor that relates with the TMS is the atmosphere since the air will be the heat sink of the TMS process.

The resulting architecture is shown in figure 5.9.

Considering then the functional behaviour of the system, the TMS shall perform two main activities:

- Absorbing heat from equipments:
  The TMS is a cooling device that aims to remove the dissipated heat from PS, PT and ES.
- Dissipate heat into the atmosphere:

The TMS shall pass the heat to the final heat sink that is the atmosphere air.

Then on the controlling side the TMS shall:

- Power its own process:

Provide the components to drive the heat removal process.

- Provide management capabilities:
  Allow the implementation of a control strategy.
- Provide a cooling mean:
  Provide a way to move the heat from the equipments to the heat sink.

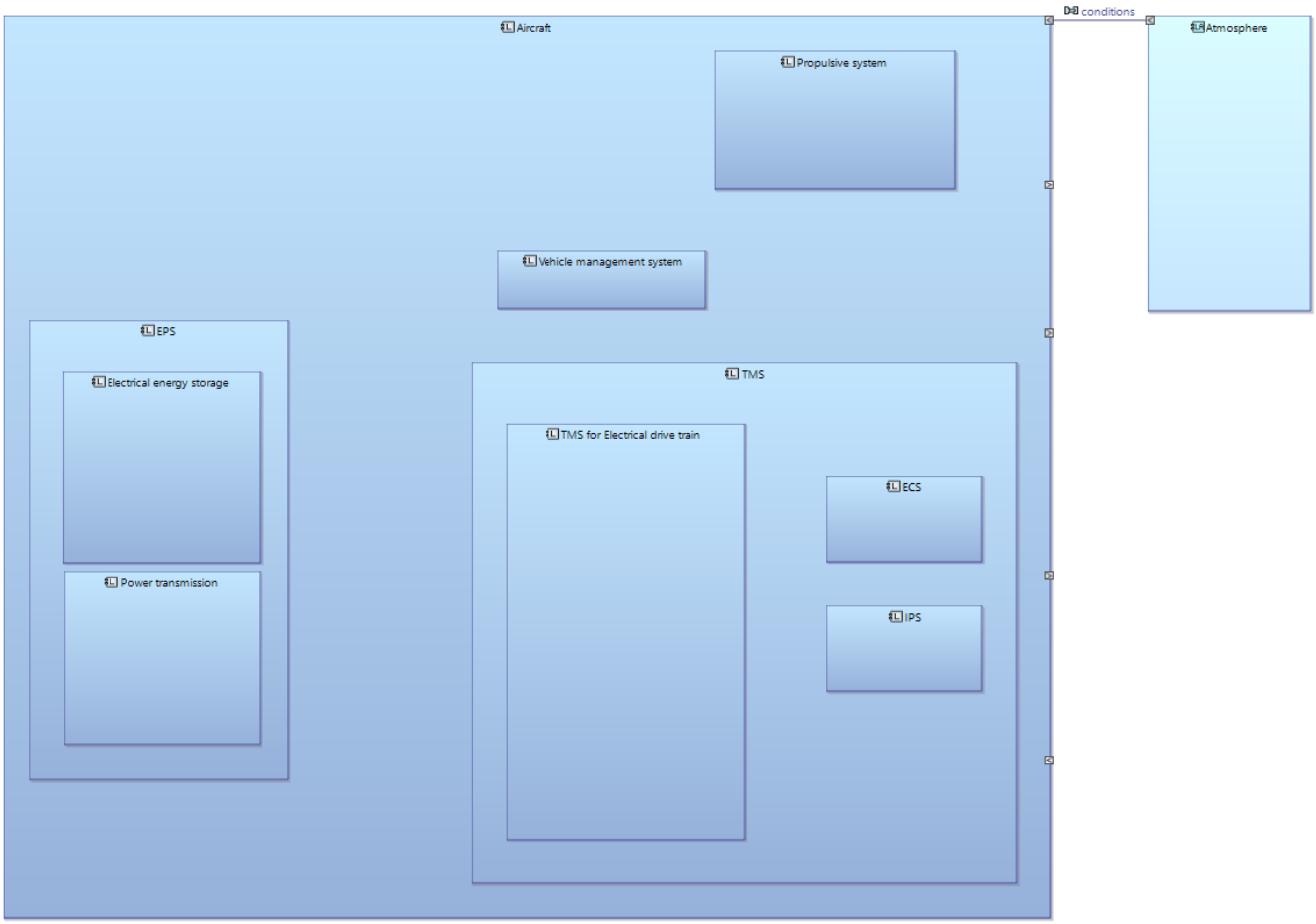

Figure 5.9: TMS logical architecture diagram (only logical elements)

At this point we change the perspective, and we try to identify which are the equipments functions that affect the TMS. The three cooled subsystems produce waste heat while performing their operation, this is the heat that interests the TMS, on the management side the subsystems will have to provide the capability to inform the TMS about their operating temperature.

Following these reasonings the function considered are:

- Propulsive system
  - Provide forward thrust:
    The electrical PS produce waste heat while operating (already created for the aircraft architecture).
  - Provide thermal information to the TMS:
    Through sensors the PS passes its thermal condition.
- Energy storage

- Store electrical power:
  The ES produce waste heat while operating.
- Provide electrical power:
  The ES produce waste heat while operating.
- Provide thermal information to the TMS:
  Through sensors the ES passes its thermal condition.

- Power transmission
  - Transmit electrical power:
    The PT produce waste heat while operating.
  - Provide thermal info to the TMS:
    Through sensors the PT passes its thermal condition.

The last created function also regards the vehicle management system that must monitor the aircraft status and thus also the TMS subsystem.

Speaking about the only external actor considered for this TMS logical level, namely the atmosphere, it relates to the TMS as it is the final heat sink, and it must absorb all the heat removed from the subsystems.

The resulting architecture with function allocation is reported in figure 5.10.

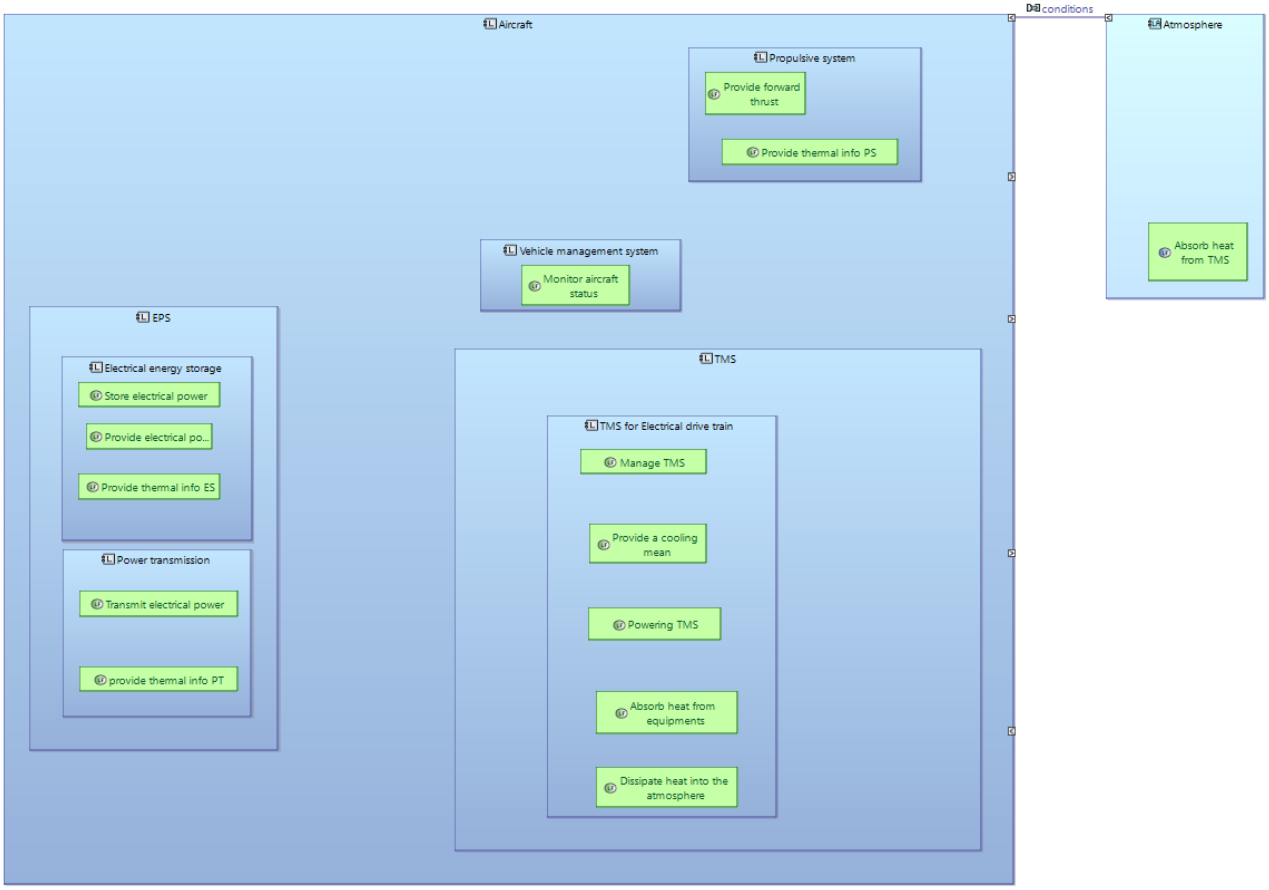

Figure 5.10: TMS logical architecture diagram with logical elements and functions allocations

The main heat chain that interests the TMS is the one that regards the heat removal process from the subsystems to the atmosphere sink (figure 5.11).

We have now to consider how are all these functions connected between each other, to do that the concept of functional chain (1) can be exploited, there are several chains to be considered and differentiating them helps clarifying how the system behaviours.

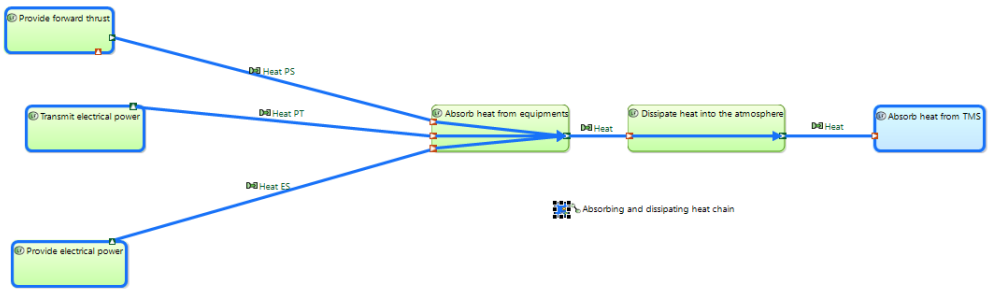

Figure 5.11: TMS absorbing and dissipating heat chain

It is now clear how the heat flows from the "source" functions to the final heat sink.

Another fundamental functional chain is the management chain that involves the functions that will impact on the control strategy of the TMS (figure 5.12).

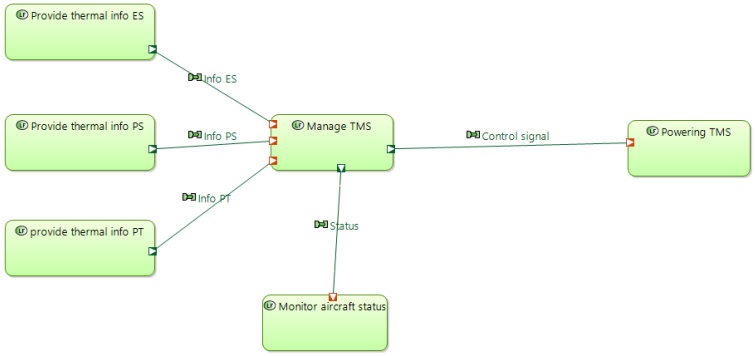

Figure 5.12: TMS management functional chain

All the equipments provide thermal information to the TMS that decides how to implement the cooling capability sending a control signal to those components that regulate the process, at the same time the TMS informs the aircraft about its status.

The last chain is not really involved in this project scope but it appears in the model and so it makes sense to make it explicit, it is the "powering" chain in which the electrical energy flows from the storage to the propulsive system, where is used to produce thrust, and to the TMS where is used to drive the heat removal process (figure 5.13).

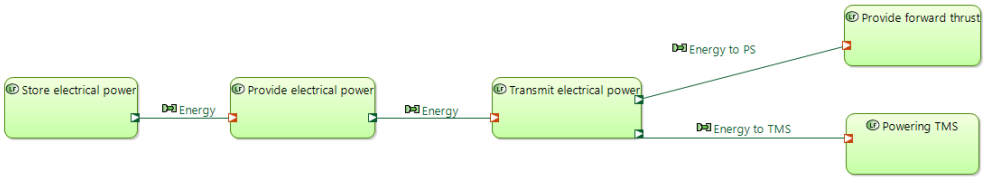

Figure 5.13: Propulsive system powering chain

The power required by the TMS is significantly lower than the power required by the electric motors, ideally the power will be transmitted to drive the first two TMS chains.

At the end if we place all the information in a single architecture diagram the result obtained for the Thermal management system is this (figure 5.14):

The chain highlighted in the diagram is the absorbing and dissipating heat chain that is the one that this project will focus more on since is the one that affects the TMS sizing and that contains many important TMS details.

The diagrams created in this paragraph are the core of the TMS model and are the base to further develop the architecture, the logical architecture diagram for the TMS is also the first step of the tool structure that this project aims to create.

It can be observed that no sequence or scenario diagrams were created to support this architecture, the aim could be not to describe the TMS cycle itself with a scenario diagram loop, that is already clear, but to associate the TMS behaviour to the mission flight phases. This would mean to depict the functional flow that characterizes a particular flight phase and eventually differentiating the TMS behaviour in different situations. At this stage of the system development however there are not sufficient information and details to define particular TMS differences between flight phases, it is known that basically the TMS is required the most when the electric thrust is needed, for instance during taxi, take-off and climb, but a controlling strategy is not implemented yet and thus during these phases it will certainly be working but there is no way to distinguish other TMS states than

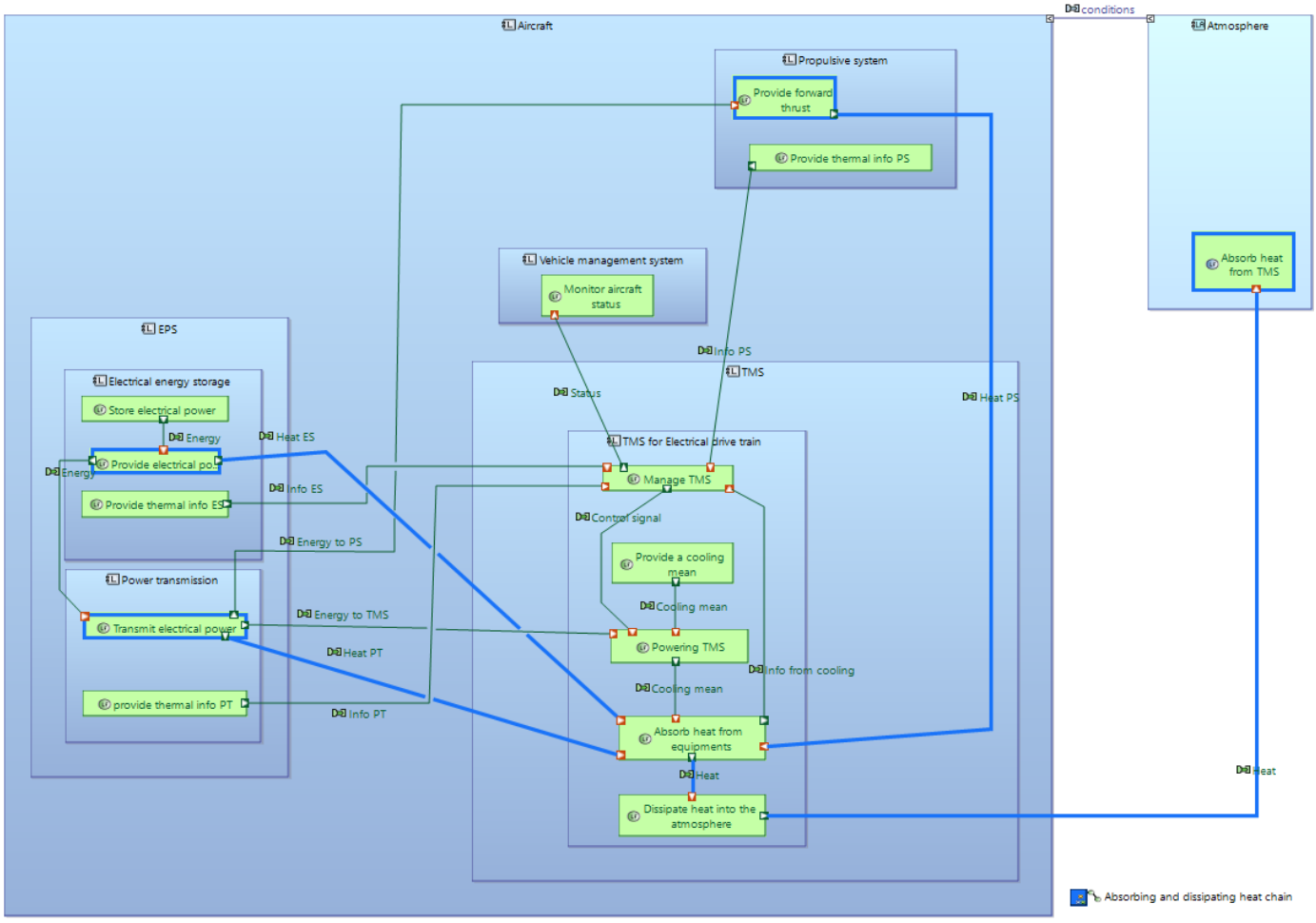

Figure 5.14: TMS logical architecture diagram

"on" and "off". Therefore, scenario diagrams would not add relevant information to the already built model.

5.2.1 Allocation of values and requirements

To be useful to the engineers the model must allow the allocation of various data inside the Capella architecture, there are several ways to add information to the Capella diagrams, but the choice must follow a method to be coherent with the model developed so far, and consistent with the TMS engineers indications.

The first way to allocate additional information into the model is the PVMT Capella add-on (section 2), this tool allows to create and associate a set of data to every Capella element present in the developed logical architecture: logical components, component exchanges, functions, functional exchanges, function and component ports, the diagram itself. This could be very useful as every element could be described some information and could carry significant values for the design process.

The main data on which the TMS is being evaluated are the heat load coming from the cooled equipments and the optimal operating temperature range required, thus it makes sense to address this information to the logical elements for PS, ES and PT. These are basically the external conditions that affect the TMS design and sizing, also in the Matlab/Simulink models these are key parameters needed to perform the preliminary systema evaluation.

Another interesting information is the TMS configuration that we want to implement and verify with the simulation.

At this point there is the need to make a critical assumption to continue developing the tool, indeed is given the possibility to associate to each of the three cooled subsystems a certain TMS configuration (option 1, 2, 3, 4). For instance there will be the possibility to state in the model that the PS is cooled with TMS option 1, the ES with option 2 and PT with option 3.

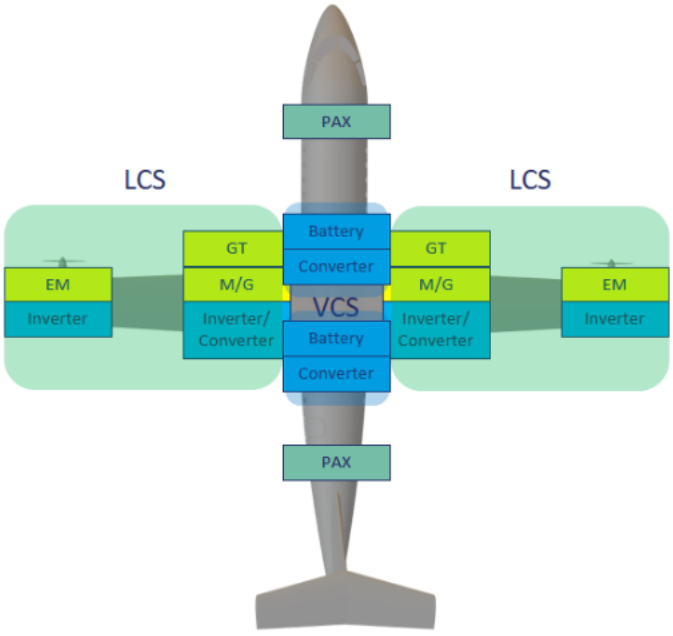

Figure 5.15: Aircraft cooling strategy

This assumption can be quite accurate if we consider that the TMS architecture is supposed to be strictly modular inside the aircraft. Referring to figure 5.15, that shows a hypothesis of aircraft TMS implementation it can be noted that different aircraft segments have different cooling strategies (liquid colling or VCS).

The TMS design team now is developing the configuration so that these different modules do not have common parts and are connected to different components. Redundancies are not yet considered but it makes sense to create a redundancy number value expecting that in the future will be used to characterize this TMS aspect.

A further consideration is that, even though different subsystems can be cooled by the same TMS, the sizing Matlab/Simulink model senses just the overall amount of heat load. Although considering multiple cooled subsystems now there is just the possibility to size the TMS with the overall heat load amount and with one reference temperature, namely the lowest or the most significant for a group of elements.

Setting in the model different TMS options for different logical elements means that ideally, we want to evaluate different TMSs tailored on each of the cooled subsystems, but since the relation between heat loads, temperatures and the Matlab results is not linear, we have to consider that the assumption could not be the most conservative one. Since this is a preliminary stage of the design process, the assumption made in this section could not badly affect the design, instead of considering a modular architecture based on aircraft subsystem's location we are basing it on functional subsystem modules, this could be almost more detailed, but we have to mind the conservative aspects.

For instance, evaluating a single TMS for two subsystems (summing the heat loads) or two different TMSs (summing the results) does not give the same result. Experimenting the model with reference values the difference is never higher then 5% of the calculated mass but the result depends on the reference temperature and on the heat load value, thus there is not the possibility to determine a constant conservative or risk factor.

After all these considerations the set of values created for the PS, ES and PT logical element is the one depicted in figure 5.16.

(Logical Component) Propulsive system

Name	Value
Heat load	
Extension_1	
Heat load	0.0 KW
T max	
Extension_1	
T max	0 °C
T min	
Extension_1	
T min	0.0 °C
TMS Configuration	
Extension_1	
Configuration	1
Extension_2	
N°	1

Figure 5.16: Propulsive system property value data set

Selecting the value box is possible to set the value for all the data considered, in the image (figure 5.16) is shown the example for the propulsive system but the layout is the same also for the energy storage and power transmission.

On the TMS side there is the need to provide a way to allocate the system description, it is assumed that this description comes from the simulation results, thus through the PVMT add-on it is created another set of values that will be filled in some way after performing the simulation.

The TMS data set is depicted in figure 5.17.

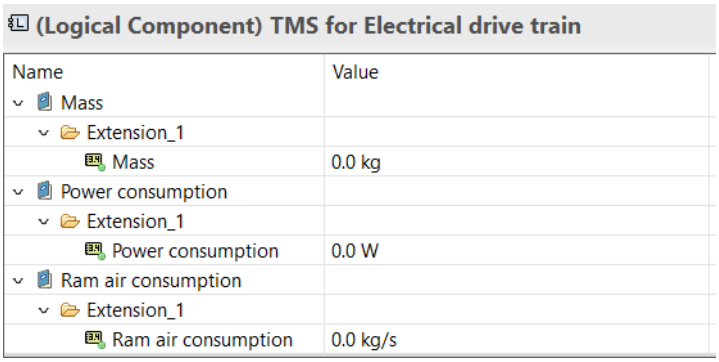

Figure 5.17: TMS property value data set

The last thing that can complete the model description at logical level is the requirement allocation, the parameters considered in the paragraph 3.9 are now translated into Capella requirement elements through the Requirement viewpoint add-on. This Capella feature is exploited to create a set of requirements that would be allocated into the TMS logical element.

Obviously, we have just the requirements for the thermal management system since the project focuses on it, but in the future the approach could be used also for other subsystems development.

The maximum thermal dissipated power is not really a requirement in this model, instead it is an input that shall be set into every cooled subsystem block and that initialise the simulation. Discussion about heat load settings will be made in the following sections.

An additional constraint that could be set as a requirement is a maximum mass limit that in the future could surely refer to a certain percentage of the aircraft MTOM.

The set of requirements created as explained is represented in the diagram as requirement elements connected to the TMS for electrical drive train logical element (figure 5.18).

The mass editing view available in Capella could be used to modify requirements values in the model (figure 5.19).

Requirement elements have a set of standard attributes, aside from them three attributes are added:

- Verification: a Boolean value that states if the requirement is satisfied.
- Description: the full requirement description.
- Value: the value associated to the requirement.

Inside the description and value attributes is placed the information reported in section 3, where possible we refer to the 2030 goal column. From this view is possible to access all the requirement

attributes and modify them when needed, for instance if during the system development the requirement value can be easily changed.

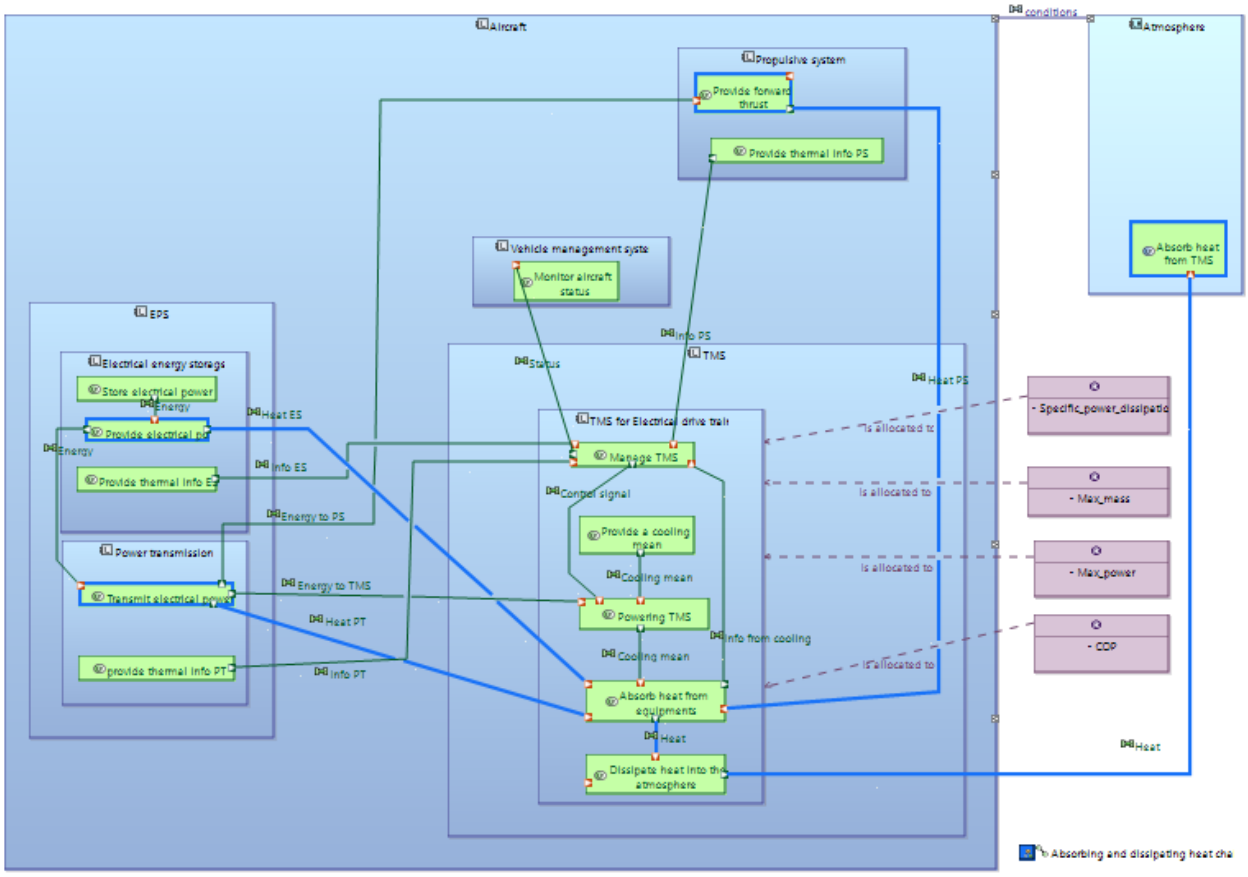

Figure 5.18: Requirement allocation into the logical architecture diagram for TMS

This paragraph completes the Capella TMS model that is one of the core components of the tool that this work aims to create, now the logical architecture, created from the aircraft perspective, focuses on the TMS, and allows the allocation of all the information that characterizes the system design. Furthermore, the allocation is coherent with the simulation models provided by the company.

After having completed this part of the project we realise that, since all the parameters, values, requirements were successfully allocated, there is no need to keep modelling also the physical architecture and EPBS layers. This is coherent with the fact that in the early stages of the system development there is not a definitive architecture at component level, therefore it makes sense to depict the TMS as a "black box". Another reason to do that is that component requirements are not available, the TMS sizing result can be compared just with requirements that refer to the overall configuration and not to single components.

	ReqIFIdentifier	ReqIFDescripti...	ReqIFLongName	ReqIFName	ReqIFPrefix	ReqIFChapterN...	ReqIFForeignID	ReqIFText	requirementTy...	Verification	Description	Value
	1		Max_mass							true	Maximum mas...	1500
	2		Max_power							true	Maximum pow...	76
	3		Specific_power...							false	Target power d...	0.5
	4		COP							true	Target Coeffici...	2.5

Figure 5.19: TMS requirements in the mass editing view

The decision is thus to stop the TMS modelling at the logical level and try to link the sizing model with this stage of the Capella model. For future works, when more component specific requirements and more detailed Matlab models will be developed, there will also be the need to create a more detailed TMS "solution" that will mean to develop the last Capella layers.

5.3 Link Matlab/Simulink models – Logical architecture

One of the main targets of this project is to create a digital interlink between the MBSE and MBD environments, therefore the goal is to implement a seamless way in which engineers could exploit simulation capabilities of an MBSE software.

For the thermal management system use case the MBSE software is Capella, and the numerical model is built in Matlab/Simulink as previously presented.

There are a few ways in which we could connect our TMS architecture with the simulation, at the end is chosen to attempt with the Python programming language. The are two reasons that drove this choice:

- First, Capella could include inside its environment a Python interpreter thanks to the Python4Capella add-on, with this feature is possible to work on our TMS model with the code. We must mind that Capella and Python releases installed on the device must be compatible, for this project are use Capella 6.0.0 and Python 3.9.
- Second, Python can include several different libraries and one of them regards the interaction with Matlab. The Matlab library for Python allows the two environments in a parallel way but requires compatible releases between them, for this work Python 3.9 and Matlab 2022a are used.

This part of the project exploits the knowledge gained during the internship (30), and a further explanation about the connection can be found in that work, now the goal is to tailor this link on the TMS.

Having said that, now we focus on the development of the code that shall manage this interlink. This connection, built in the P4C environment, must perform some determined steps to achieve the goal:

- Retrieve from the Capella model the values needed for the simulation.
- Send these values into the Matlab workspace.
- Run the Matlab pre-processing script.
- Run the Simulink model.
- Run the Matlab post-processing script.
- Retrieve results from the Matlab workspace.
- Retrieve the requirements allocated in the model.
- Compare simulation results with requirements.
- Place the all the results in the Capella model.

Following these steps the P4C code first gets the values regarding the heat load and the lower limit of the operating temperature range (most demanding condition) for the PS, ES and PT.

Then using the matlab.engine library these data are sent into the Matlab workspace, the pre-processing scripts are slightly modified from the base version provided by the company as now the heat load and reference temperature are already defined through P4C and thus there is not the need to declare them in the Matlab script.

To make the Simulink model run automatically we add at the beginning of the post-processing script the line of code that runs a Simulink file from Matlab ("*sim*" function), the same result could be obtained placing the run function at the end of the post processing script.

In this way pre-processing and post-processing scripts run in sequence both launched from the P4C environment thanks to the proper matlab.engine function, once Matlab stops running the P4C code gets from the Matlab workspace the values for the overall mass, ram air consumption, power consumption just accessing their workspace variables. The post-processing script is slightly modified to output exactly the overall amount of the parameters of interest.

In the previous paragraphs it was mentioned that for every cooled subsystem a different TMS option could be set, indeed the code is built to run the respective Matlab/Simulink model depending on the "configuration" input. Thus, the code runs three times (three cooled subsystems), and every run could use one of the four model options depending on the "configuration" number.

If in the future other Matlab/Simulink models will be created there would be the possibility to link these new ones in the same way.

While running the code retrieves the results of the three simulations, prints in the console the results of each iteration and at the end sums them, at this point stored in P4C there are the overall results of the TMS summing the TMS estimations for each subsystem.

The script is written to run a simulation for every aircraft subsystem that at logical level has a configuration attribute different to "0", this because in possible future works it could be interesting to consider also other segments to be cooled by the TMS, therefore for a preliminary evaluation of the impact on the cooling sizing it will be sufficient to give to the new considered subsystem the values of operating temperature, heat load produced and cooling option chosen different from "0".

Now the requirement values are retrieved from the model and compared with the results, in some cases a simple calculation is needed to obtain a certain parameter from the results available (requirements definitions in section 3).

An interesting choice that is made at this point, according with the TMS engineers, is to create not just a variable that tells if the requirement is satisfied or not but also to output a percentage of how close the result is from the limit/target value. The percentage is calculated as:

$$result\ (\%) = \frac{simulation\ result}{requirement\ value} * 100 \tag{5.1}$$

This procedure cannot be called a proper requirement verification because it is just a first comparison of a sizing model with a requirement developed in an early stage, anyway the comparison gives interesting information that could be useful to the engineers during the development activity, this is also why the static Matlab/Simulink model was developed by the company.

The last thing that the code must perform is to display the results of the simulation and the requirement check inside the Capella model in very easy-to-read way. In the previous paragraph, with the PVMT add-on was created a set of properties allocated in the TMS logical element, these values are set directly with P4C allocating the mass, ram air and power consumption results. The idea is to expand this set adding all the requirement comparison results, both the "boolean" and the percentage values, in the same way the PVMT view is filled from the P4C environment with the code.

The resulting property value view of the TMS logical element is visible in figure 5.19.

This table is filled entirely from the code, the idea is to make the results available all in a single place without having to check either the code or its output in the console. The code at the end modifies the model as it changes the description of the TMS element, to perform this operation with P4C we must state in the code that we are performing a "transaction", in the other cases the code would not change the model allocating values.

(Logical Component) TMS for Electrical drive train

Name	Value
COP_req	
Extension_1	
COP_req	0.0 %
Extension_2	
COP_req_bool	...
Mass	
Extension_1	
Mass	0.0 kg
Mass_req	
Extension_1	
Mass_req	0.0 %
Extension_2	
Mass_req_bool	...
Power consumption	
Extension_1	
Power consumption	0.0 W
Power_req	
Extension_1	
Power_req	0.0 %
Extension_2	
Power_req_bool	...
Ram air consumption	
Extension_1	
Ram air consumption	0.0 kg/s
SPD_req	
Extension_1	
SPD_req	0.0 %
Extension_2	
SPD_req_bool	...

Figure 5.19: Updated TMS property value data set

As it can be observed the set is formed by:

- Mass: mass result
- Ram air consumption: ram air consumption result
- Power consumption: power consumption result
- Mass_req: percentage of mass compared to the requirement constrain
- Mass_req_bool: boolean value for mass requirement verification

- Power_req: percentage of power consumption compared to the requirement constrain
- Power_req_bool: boolean value for power consumption requirement verification
- COP_req: percentage of COP compared to the requirement target
- COP_req_bool: boolean value for COP requirement verification
- SPD_req: percentage of SPD compared to the requirement target
- SPD_req_bool: boolean value for SPD requirement verification

There were other ways to report the result information into the model, but this choice was made basing on a trade-off between what was feasible with P4C and creating a clear result representation in the MBSE model.

For developing purposes, the code also prints in the console some information, these are: updates on the run status, the summary of the characteristics of every subsystem cooled, and all the results that are also placed in the model.

The full Python4Capella script is available in Appendix III, this paragraph has reported the explanation of what brought to that result.

Similarly, to the internship to build the code were used the function provided by Python4Capella in the "simplified API" and "utilities" libraries. Many code issues were solved thanks to the discussion that took place at (31), when needed some new P4C function were created. This worked also helped the community to improve the add-on itself and some Capella features, since some aspects of the tool are not explored yet, this was also a reason that drove the activity of building the digital interlink: not everything in the Capella environment was accessible with P4C and some features are still under development.

To create this code, it was frequently needed a learning phase regarding some Python and P4C features (32), (33), (34), this activity was not reported in this paragraph as it is not in the scope of the project, the information written here refer to the approach and the methodology used to write this digital connection.

Note that, as said in the previous section, it is not a scope of this project to either discuss or modify the model provided by Ebraer Research and Technology Europe - Airholding S.A., therefore changes were made just to make the models compatible with the digital connection architecture.

At this point of the project we could state that the digital "tool" is completed: the TMS Capella model was built from an aircraft perspective, the model was built with a level of detail that allows to allocate all interesting elements for this stage of the development, the Matlab/Simulink models are connected in a seamless way that requires just to run the P4C code, a preliminary requirement check is performed by the code and all results are available inside the available logical diagrams.

5.4 Phase one results

The aim of this section is to show how the tool could be exploited during engineering processes, a sort of tutorial will be crated explaining how to use it and who can benefit more from this structure. Also, a numerical example will be reported showing at the end how results could be interpreted and how the digital tool boosts the TMS project development. These two activities will be reported in parallel as the numerical example will be used to explain the tool functioning.

Before the tutorial it is useful to remark at which stage of the TMS development process this work is collocated, in section 3 the flowchart of the engineering activities was created, and this project wants to help during the first unconstrained iteration phase. Now there is a new tool that could be used in this phase thus changing some of the previous considered activities.

The new flowchart representing the engineering workflow for the first unconstrained iteration is this (Figure 5.20):

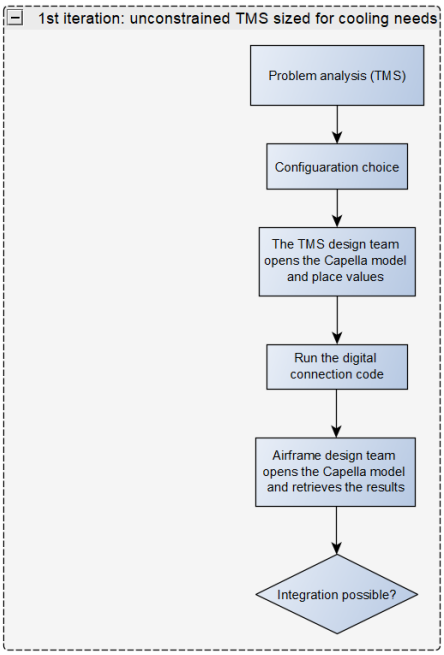

Figure 5.20: New engineering workflow flowchart

We must keep in mind that the TMS Capella model aims to be "shareable" between all the teams and engineers working on the system development.

Now when the TMS design team has evaluated some constraints and condition and the static Matlab/Simulink models are available, the engineers could place all the values and requirements describing the model that allow to initialise the simulation. These are temperature ranges, heat loads, configuration option for all the three cooled subsystems.

Temperature ranges are chosen from optimal operating temperature ranges presented in section 3, for the heat loads instead is considered the table in Appendix IV, the table reports the dissipated power for the cooled components considered in this use case, it assumed that all the dissipated power is formed by waste heat that the TMS must remove.

As presented in the requirements table 3.7 there is a target value for the maximum dissipated power, this value is not really related to the sum of all the dissipated powers in a single flight phase that could be evaluated from the table in Appendix IV. The reason is that in Appendix IV the values come from statistical research on similar architectures and components, these values are constantly updated basing on the system development, so it does not make sense to rigidly consider just the values in the table. For instance, in Appendix IV the resulting most demanding condition is the take-off phase, but after some weeks that this table was provided the TMS engineers stated that new estimations were saying that the most demanding phase was the end-of-climb segment since all the electrical thrust is required at high altitudes in climb condition. Also, the taxi phase is way more demanding then it could seem as the target is to perform a full electric-powered taxi phase, thus

putting electrical components under pressure. It is then clear that the heat loads values depend on the most recent evaluations and expert opinions.

Another assumption that was made to make the table in Appendix IV compatible with the model is that batteries are supposed to be the only energy storage component; converters, inverters and distribution refer to power transmission, electric motors and electric boosters are propulsive system components. Therefore, the sum of dissipated power for each group is allocated into its own logical component.

Despite that it must be noted is the approach with which for every flight phase and more in general for every condition there must be considered a different heat load value given by a different behaviour of the cooled component.

In this numerical example the PS, ES and PT thermal characteristics inserted in the model are shown in figures 5.21, 5.22, 5.23.

(Logical Component) Propulsive system

Name	Value
Heat load	
Extension_1	
Heat load	50.0 KW
T max	
Extension_1	
T max	200 °C
T min	
Extension_1	
T min	0.0 °C
TMS Configuration	
Extension_1	
Configuration	1
Extension_2	
N°	1

Figure 5.21: PS data set

(Logical Component) Electrical energy storage

Name	Value
Heat load	
Extension_1	
Heat load	90.0 KW
T max	
Extension_1	
T max	30 °C
T min	
Extension_1	
T min	20.0 °C
TMS Configuration	
Extension_1	
Configuration	1
Extension_2	
N°	1

Figure 5.22: ES data set

(Logical Component) Power transmission

Name	Value
Heat load	
Extension_1	
Heat load	50.0 KW
T max	
Extension_1	
T max	120 °C
T min	
Extension_1	
T min	80.0 °C
TMS Configuration	
Extension_1	
Configuration	2
Extension_2	
N°	1

Figure 5.23: PT data set

For this example, TMS option 1 is chosen for PS and ES while option 2 is selected for PT, the number of redundancies is always 1 as explained in section 3 and 4, the redundancy number in the future will multiply the result to obtain the overall configuration result.

Requirements are the same shown in table 3.7 and in the requirement visualization (figure 5.19), the mass requirement value is a hypothetical value chosen based on previously performed simulations, realistically it will be set as soon as it will be available a certain percentage of the MTOM.

It was chosen to set the heat loads in the way that their sum is 190 KW, this is exactly the maximum value in 3.7 table, this choice was made to verify that condition that is also coherent with a take-off/climb condition in Appendix IV table.

Once that this information is allocated in the TMS logical architecture diagram it is now possible to run the code that connects the simulation and crates the "digital bridge" between Capella and Matlab/Simulink. An interesting feature of the tool crated in chapter 5 is that the engineers do not have to read the code and understand how it is made, there is not almost the need to check what the console prints as all the main results are shown in the TMS data set. It is possible to run it directly from the Capella project explorer right-clicking on the *matlab_connection* script and running it as an EASE Script (figure 5.24):

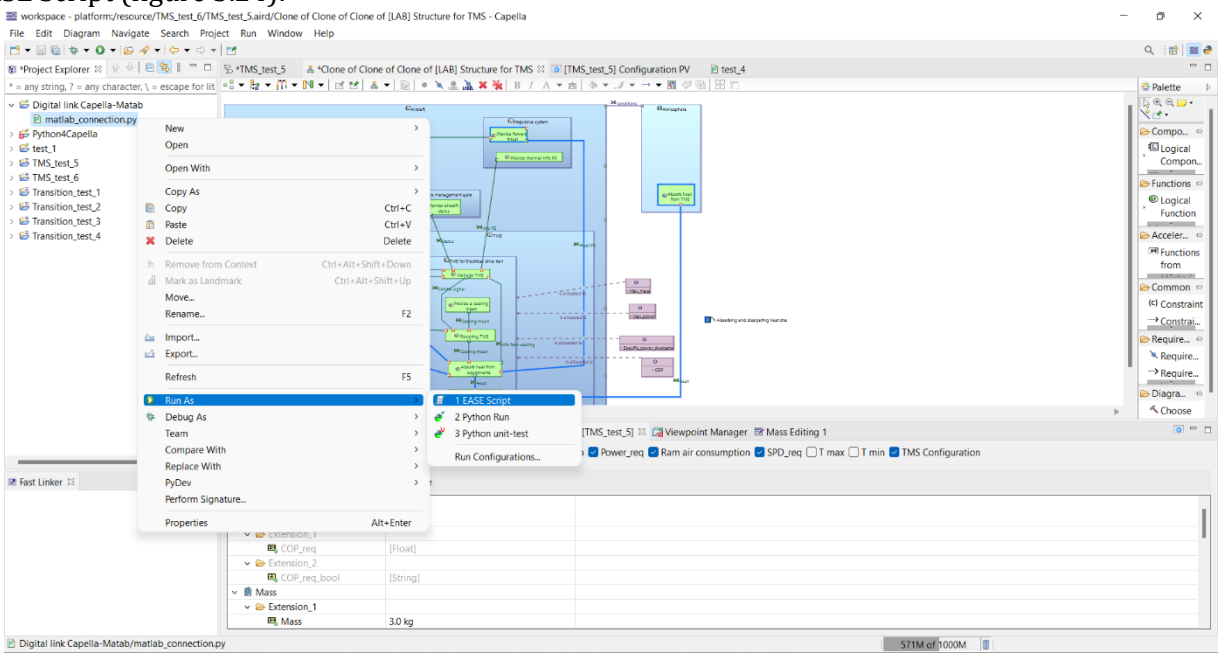

Figure 5.24: Running the code illustration

In this example for instance after having clicked the run button and waited for the run to complete the results in the TMS data set are shown in figure 5.25.

The code could require about thirty seconds to run completely, depending on the device, because it could take some time to open the Matlab environment from Python.

(Logical Component) TMS for Electrical drive train

Name	Value
COP_req	
Extension_1	
COP_req	111.33 %
Extension_2	
COP_req_bool	True
Mass	
Extension_1	
Mass	1232.98 kg
Power consumption	
Extension_1	
Power consumption	68268.19 W
Power_req	
Extension_1	
Power_req	94.82 %
Extension_2	
Power_req_bool	False
Ram air consumption	
Extension_1	
Ram air consumption	5.19 kg/s
SPD_req	
Extension_1	
SPD_req	30.82 %
Extension_2	
SPD_req_bool	True

Figure 5.25: Simulation and requirement comparison results in the TMS logical element data set

In this example for instance the evaluated mass and power consumption are under the requirement limit and how far they are from that is shown with the percentage, the Boolean value says “True” as the requirement constrain is respected. The COP result satisfy the target value in the requirements as we see a percentage slightly higher than 100%, coherently the Boolean verification value says “True”. Otherwise, the specific power dissipation reaches just 30% of the requirement target value, thus the requirement is not fulfilled and the Boolean value states “False”.

Now the engineering workflow could easily keep going as the airframe design team engineers could freely access this model checking the results obtained by the preliminary TMS evaluation, the full process is very lean and fast because once the TMS team has performed its evaluation it has just to allocate some values, run the connection and the main results are immediately available to the airframe engineers and the discussion about integration issue could start. This is a sort of platform where all the key TMS information are allocated and are automatically placed after the simulation avoiding the risk of misunderstandings and data losses.

For instance, speaking about this example, it could be stated that, given these requirements, the TMS configuration as evaluated does not weight too much and does not consume too much power. The coefficient of performance satisfies the requirement and so the proportion of electric power used to the thermal power removed is adequate with this TMS option set. On the other side the specific power dissipation is far from the target value, this means that the “density” of the thermal power removed is too low, the TMS in this configuration requires an excessive mass for a certain thermal power to absorb, this in the aircraft could translate into excessive masses and volumes if a high thermal power is expected to be removed.

Having said that the work can for instance be directed to searching for a solution that could absorb more thermal load with the same mass, attempts could be made changing the TMS configuration by selecting different cooling options. Another way is to look for a technology improvement in components and materials that could benefit the TMS layout, anyway this is the kind of discussion that could follow in the future design loops in which a solution will iteratively be found.

For completeness it is reported also what the code prints in the console:

```
Start

Airframe

Propulsive system
  Configuration 1
  N° 1
  Heat load 50.0 KW
  T max 473.0 K°
  T min 273.0 K°
  Energy Balance of Liquid cooling verified
  Energy Balance of VCS verified
  Mass = 339.46 kg, Ram air consumption = 1.68 kg/s, Power consumption =  21891.16 W

Communication system

Navigation system

FCS

Vehicle management system

Landing gear
EPS

Electrical energy storage
  Configuration 1
  N° 1
  Heat load 90.0 KW
  T max 303.0 K°
  T min 293.0 K°
  Energy Balance of Liquid cooling verified
  Energy Balance of VCS verified
  Mass = 538.89 kg, Ram air consumption = 3.03 kg/s, Power consumption = 39460.46 W

Power transmission
  Configuration 2
  N° 1
  Heat load 50.0 KW
  T max 393.0 K°
  T min 353.0 K°
  Energy Balance of Liquid cooling verified
  Energy Balance of VCS verified
  Mass = 354.62 kg, Ram air consumption = 0.48 kg/s, Power consumption = 6916.57 W

Hydraulic system

Pneumatic system

Survellance and identification system

Fuel system

TMS

TMS for Electrical drive train

IPS
```

```
ECS

----

Total mass = 1232.98 kg
Total ram air consumption = 5.19 kg/s
Total power consumption = 68.27 KW
Total heat load dissipated = 190.0 KW
Specific power dissipation = 0.154 KW/kg
COP = 2.783

Mass obtained is 82.199% to the Maximum Mass
Power obtained is 94.817% to the Maximum Power
Specific power dissipation obtained is 30.82% to the target Specific power dissipation
COP obtained is 111.326% to the COP target

End
```

For every logical subsystem considered in the TMS cooling configuration there is a brief summary of thermal characteristics, then the Matlab output is printed with the result for the single Matlab/Simulink model run for that specific subsystem (hypothetical subsystem specifically evaluated for that case).

In the last part the results that are also reported in the model are printed out. Since considering the result for the singles "modules" of the configuration seems not to be interesting to the TMS engineers at this early stage of the development that information was not reported in any PVMT variable, but if we want to check how every module affect the overall result it is possible to easily look at the results in the console.

This section concludes the core of this project, starting from the problem definition the work came through the building process of the "digital tool" that includes the TMS MBSE model from an aircraft perspective, and the connection with the sizing Matlab/Simulink models. The goal seems to have been achieved as this tool could be a very useful framework that could help engineers throughout their activities boosting the discussion between design teams, furthermore this structure and approach could be translated for the other aircraft subsystems crating a full aircraft model where all the subsystems logical elements have a seamless connection with their own simulation models.

The result at the end satisfies the engineers and gives useful insights about the ongoing project.

6 Building the digital tool – Phase two

This chapter concerns about topics that were investigated after the creation of the digital link, there were two main research paths that could be followed at this point:

- Capella physical architecture layer
- Scenarios and system states description

The first one aims to investigate if a hypothetical solution proposal for the TMS at component level could create in the Capella environment and eventually if it could be useful for the system development.

The second research has the scope to identify a possible methodology that could associate flight phases with a certain TMS scenario or system state.

6.1 TMS physical architecture layer

Starting from the TMS model developed in the logical Capella level we want now to "open" the TMS logical block detailing how this system will be built. This is the main scope of this Capella layer, if the system analysis aims to define what the system has to do, in the physical architecture layer we want to propose a "solution" that specifies subsystem physical components and interfaces.

The physical layer development process starts from the main diagram of the previous layer that is the logical architecture diagram in figure 5.18. At this point we exploit the "system to subsystem transition" Capella feature, this add-on allows to create a separate model in which the modelling process could follow, in this way for each TMS layout proposal we could create a different transition model, but all the transition would be related to the original main branch of the model that has stopped at logical level.

For instance, a possible solution is the one that refers to figure 5.15, in that case we assumed that the energy storage components are cooled by an option 1 TMS and the propulsive system and power transmission are cooled by an option 3 TMS.

In this example it is considered just the "absorbing and dissipating heat chain" and thus in the physical architecture are created just those physical nodes directly involved in the heat removal process.

Figure 6.1 shows a possible physical architecture.

Coherently with what said above the batteries are cooled through a liquid cycle combined with VCS, that means that the involved components are the liquid refrigerant that removes the heat in first place, the pump that drives the liquid, the evaporator that transfers the heat to the changing phase gas, the compressor that moves the gas and the condenser that transfers the heat to the final heat sink that is the air in the atmosphere.

On the other hand, all the PS and PT components mentioned in section 5 are cooled just with a liquid cooling cycle that similarly has the liquid refrigerant and a pump but then the ram air heat exchanger passes the heat directly to the air.

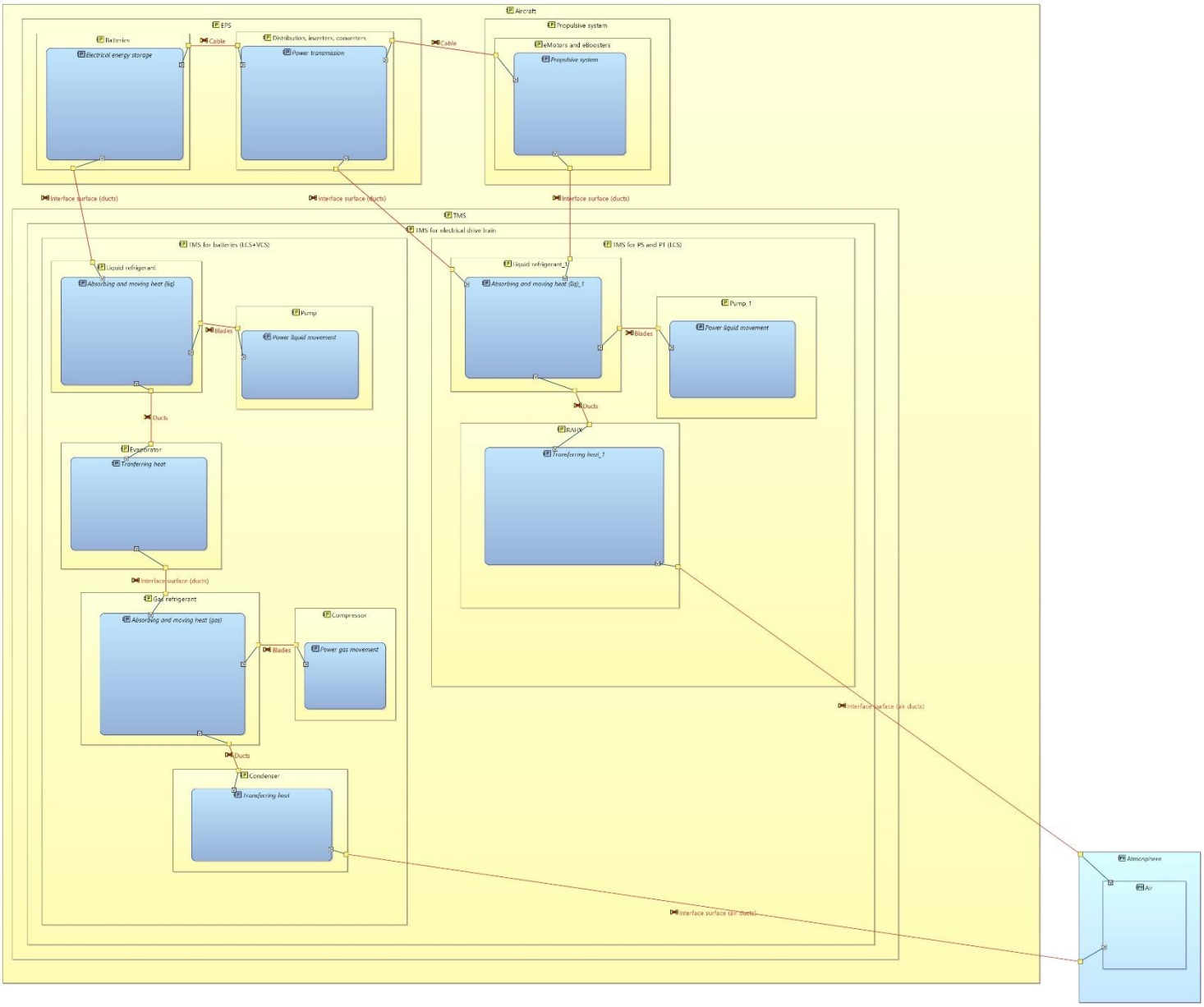

Figure 6.1: TMS physical architecture diagram (only nodes PC and behaviour PC)

In this framework it is assumed that the ducts are the components that connect every heat removal stage as through ducts both EGW and R314 move allowing the heat to be transferred from the source component to the heat sink, ducts also are a subcomponent of physical elements that transfer heat from one mean to another because they form an interface surface.

The parts that allow the pump and the compressor to transfer the mechanical energy to the fluid are their blades, thus these elements are depicted as connections.

On the functional side there is the need to make even more explicit those functions considered at logical level, this because we have now to specify the functional meaning for every existing node PC. The chain considered at logical level will be split into two different functional paths one for each TMS option (figure 6.2).

The two different paths describe how the heat moves through the components and how every component participates to this process:

- Liquid refrigerant
  - Absorb waste heat
  - Move the heat
- Pump
  - Move the liquid
- Evaporator
  - Transfer the heat from the liquid to the phase changing gas
- Gas refrigerant
  - Absorb the heat
  - Move the heat
- Compressor
  - Move the phase changing gas
- Condenser
  - Transfer the heat to the air heat sink

In the functional diagram it appears also the chain that functionally transfers the electrical energy from the storage to the user, namely the electrical motors.

The resulting physical architecture diagram with function allocation is depicted in figure 6.3.

This is just an example of how a hypothetical physical layer could result for this use case, if for some reasons we want to propose another solution, for instance implementing option 2 for batteries and option 4 for the other components, it is possible to create a new transition in which a new physical architecture will be developed, figure 6.4 shows the physical architecture diagram for this new example.

In this second transition there are the skin heat exchangers components that carry the function to create an additional path that allow the heat to move into the atmosphere.

We must consider that although we have created a more detailed architecture, we have also to assess the model utility from a system perspective. This model allows to describe the system to the component level since is possible to insert information about specific physical components, physical constraints, interfaces, component requirements. However, at this stage of the TMS development there are not yet this kind of details, also there is not a dynamic model that could simulate the system behaviour evaluating temperatures and components results, thus, even if component requirements were available, there would not be the possibility to verify them.

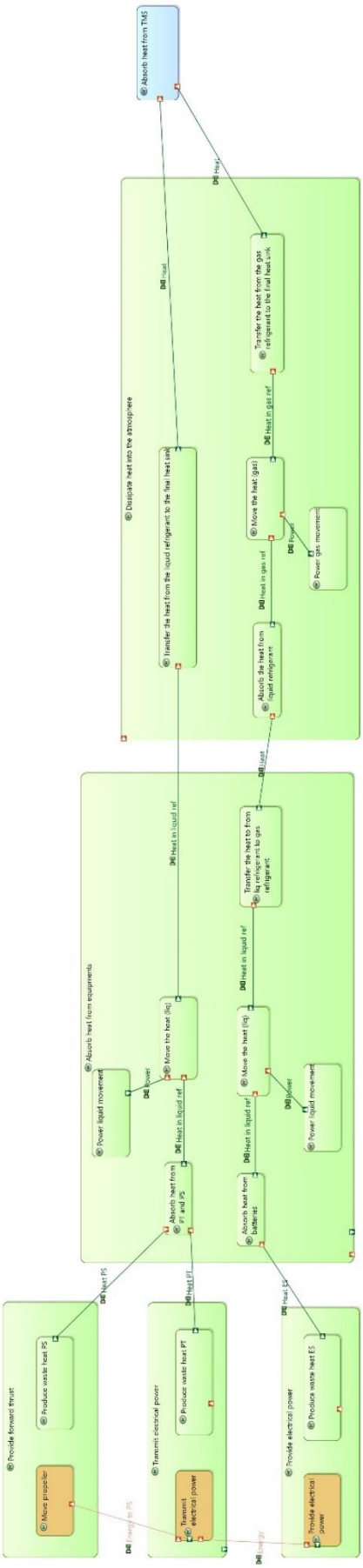

Figure 6.2: TMS physical functional dataflow diagram (transition 1)

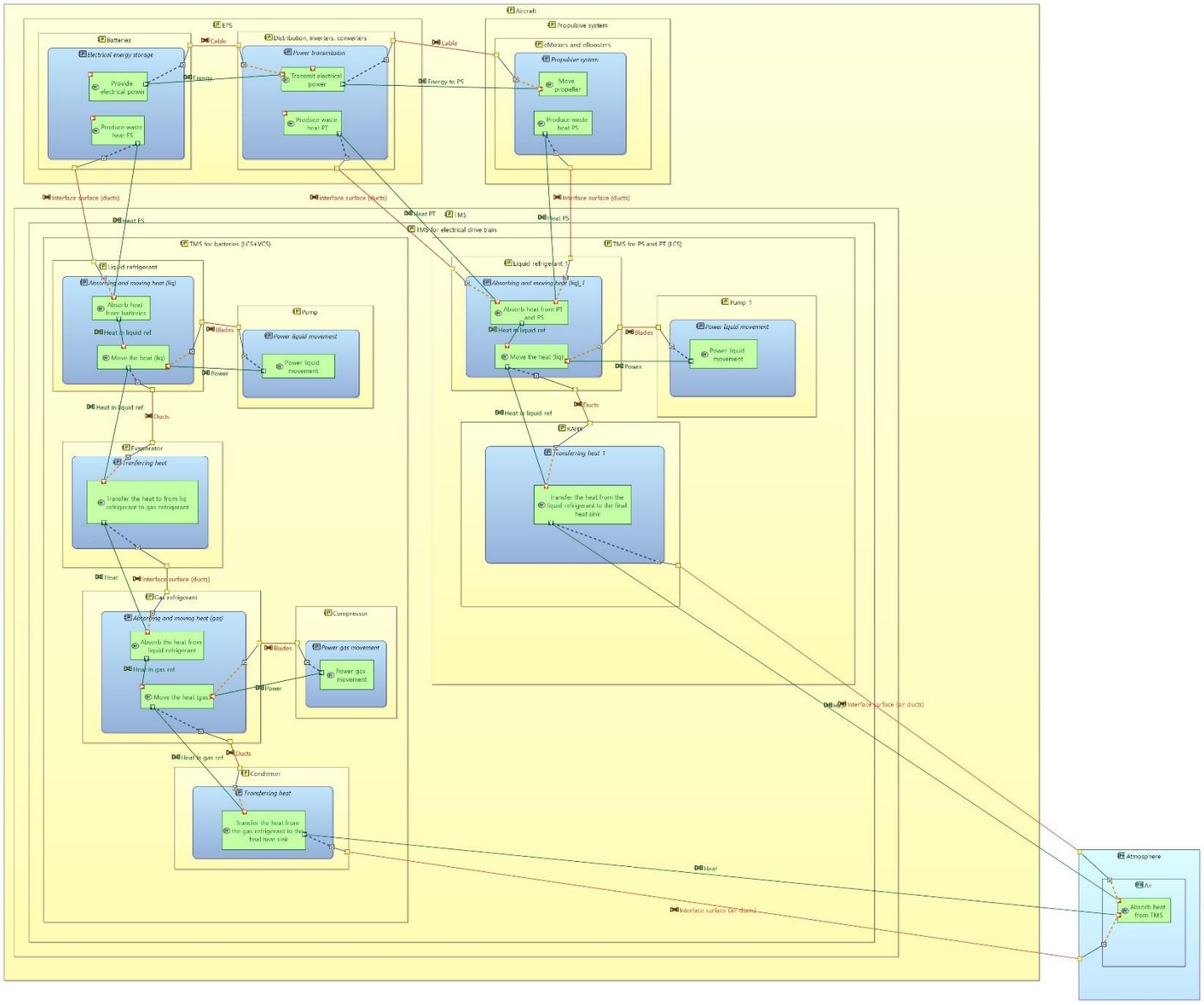

Figure 6.3: TMS physical architecture diagram (transition 1)

All these reasons brought to the conclusion that the creating the physical architecture layer could be a good exercise to understand the system behaviour considering TMS subcomponents.

Once that the system development will be at an advanced stage this physical framework could be used to allocate data and information as it was done with the logical level for this project. Furthermore, when a dynamic simulation model will be developed including component details it will be feasible to build a digital connection with these models that could link directly the physical layer, this would mean to create a framework and a digital connection tool identical to the once built for this work but transferred at component level as the system will be more detailed.

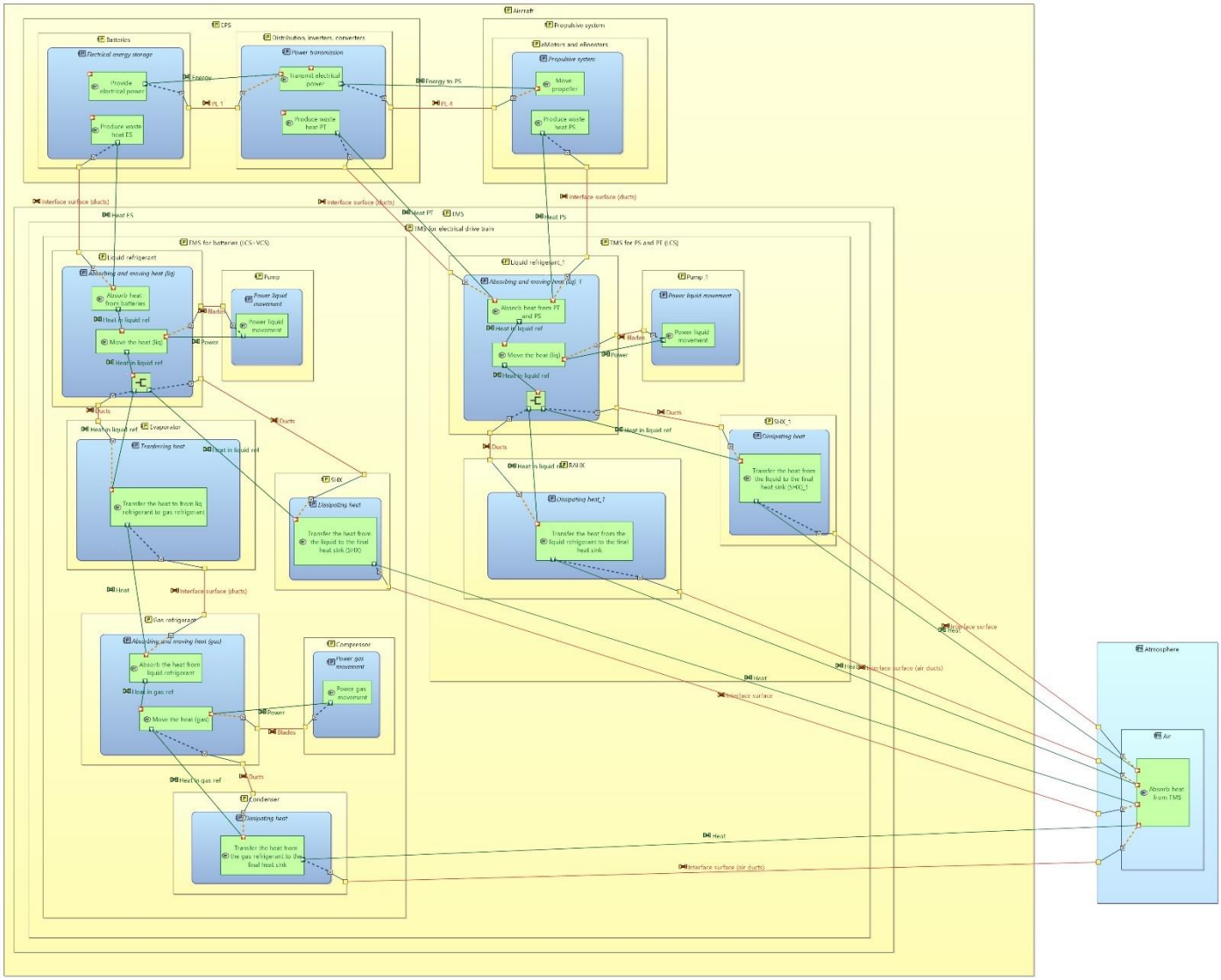

Figure 6.4: TMS physical architecture diagram (transition 2)

6.2 TMS states and scenarios

In chapter 5 was pointed out the need of specifying the TMS behaviour throughout the reference mission, as previously said we want to characterize the different behaviour of the system in different flight phases.

The goal is to create a solid reference accessible on the model that could help the engineers to decide what critical phases of the system to simulate and to find the sizing condition. Indeed during the system development process there will not be a single run of the "digital tool" to evaluate the system, more realistically a set of different conditions, brought from statistical analysis on the available technology, would be tried to evaluate the system preliminary sizing.

From the discussion with the engineers emerged that the TMS is needed mainly when high-level of electric thrust is required, these conditions can be found in different flight phases:

- Taxi: for sustainability goals the target for the aircraft is to perform a full-electric taxi phase.
- Take-off: maximum thrust required.
- Climb: as the aircraft increases its altitude the air density lowers causing a decrease of thrust produced with turbo-engines that must be compensated with electrical motors.
- Go-around: similarly to the climb phase the aircraft needs thrust to get again in a climb condition.

Particular attention must be placed to the end-of-climb segment, this because all the electrical power train suffers of overheating when the electric thrust is required for a long time. Thus the longer is the mission segment then more heat will have to be removed at the end of that phase.

If for instance, we consider figure 6.5 where batteries resistive losses are plotted along the mission profile with the total thrust required is possible to

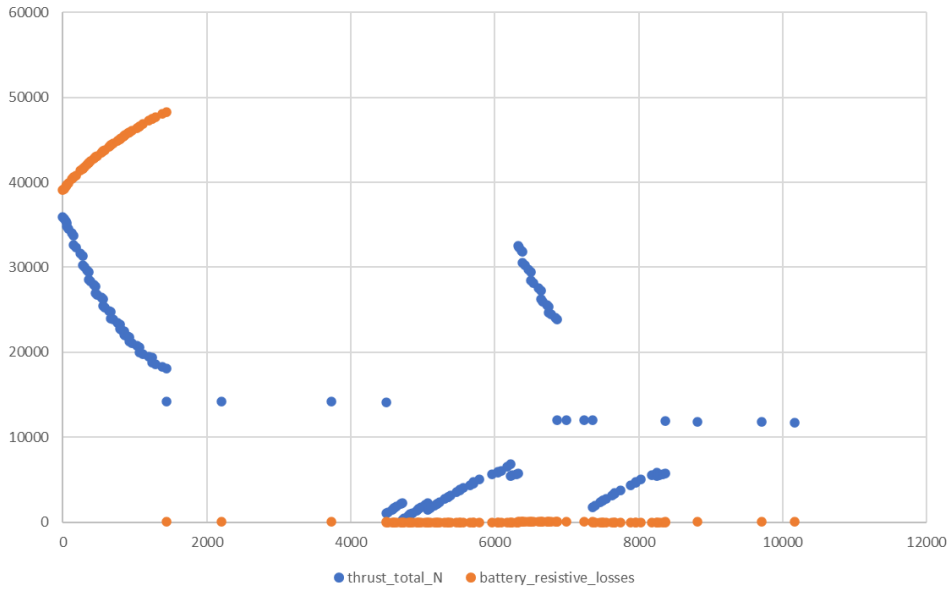

Figure 6.5: Batteries resistive losses and total thrust along the mission

This graph does not consider the taxi and take-off phase where the electric thrust is required to propel the aircraft. The data used to create the chart were picked from the table in Appendix V, this table was available just at the end of the project and the data come from a first evaluation of a dynamic simulation model that considers technologies like the one that will be implemented in the FutPrint50 aircraft.

Therefore, starting from the taxi phase to the end-of-climb the TMS is working to remove the heat, the heat load peak is reached at the end of this long period. During go-around is required again to have high-levels of thrust but since there is less operating time for the electrical drive train the amount of waste heat produced is smaller.

This graph shows clearly when there is the need to implement a cooling system for the electrical drive train, this is the first step in the development of a control strategy. It could also be noted from the table in Appendix V that even if batteries are not providing power there is a small amount of heat produced also in a “stan-by” battery state.

6.2.1 TMS logical architecture states and modes

Given the reported information is possible to create in the Capella model some diagrams depicting the different states of the considered subsystems. These are the mode and state machine diagrams (MSM) on which the system is considered from the initial “building” to the final “dismissed” states.

In Capella is available the description of an element condition both in terms of states and modes, the main difference between these two is that a state is an internal condition that defines how the element is behaving, on the other hand a mode is a condition imposed by the external context and environment in which an element operates.

The diagrams that will be created in the following work are defined basing on the internal behaviour and thus they are state diagrams.

First are create the states diagrams for the cooled subsystems of the electrical drive train (figures 6.6, 6.7, 6.8):

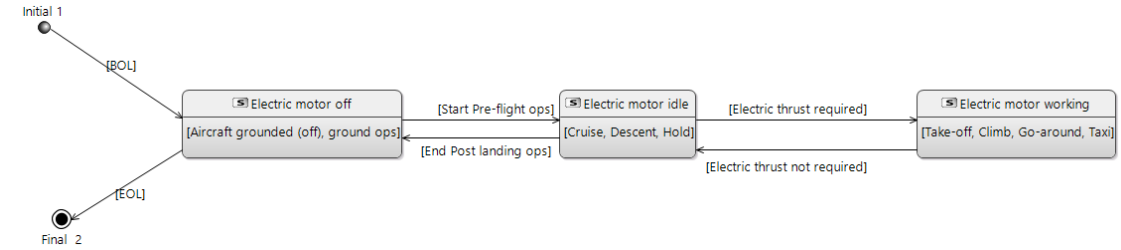

Figure 6.6: Propulsive system MSM diagram

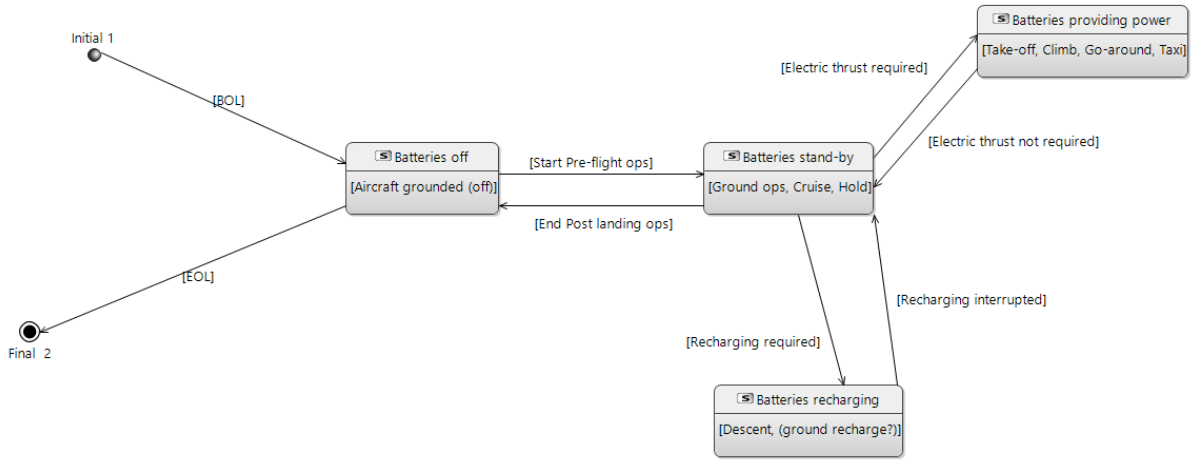

Figure 6.7: Energy storage MSM diagram

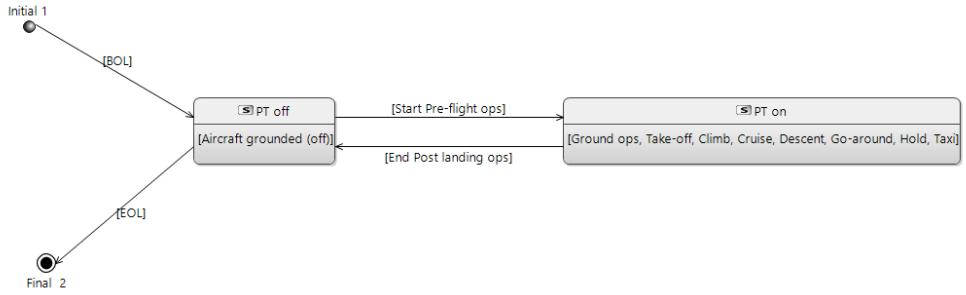

Figure 6.8: Power transmission MSM diagram

These diagrams have blocks that represents a system state, the connections are basically triggers that make the system change state, in these diagrams triggers are also interpreted as a change in aircraft operations where it is assumed that the system would receive a signal from either the pilot or the flight computer to change its state since an action is required. Then, since every state blocks allows a description, the different flight phases are associated to each state.

For the electric motors we consider that they are turned off during ground operations, an idle mode is created for all those flight phases in which the electrical drive train is in a stand-by state where it is turned on but not providing power, these phases are for instance cruise descent and hold. During all the other already mentioned phases the electric thrust is assumed to be required and so the electric motors are providing thrust.

The diagram for batteries is coherent with the one for the propulsive system since this one is the system that commands when he needs the power, in these flight phases the batteries are providing the electrical power needed to produce thrust and dissipating heat. In the other phases the batteries could be in a stand-by state where, as said before, they still produce waste heat but in a smaller amount. It is also assumed that a possible recharge state could be representative of a descent condition where batteries could be recharged by the GT engines. During ground operations it is hypothesized that batteries are turned off, anyway it makes sense to expect that the aircraft will be connected to a GPU that would recharge the batteries, this process would certainly need to be cooled but it is not known yet if with an external cooling mean or with the TMS considered in this project, future aircraft developments will specify that.

For the power transmission we just differentiate the two main conditions: the one where the electric line is not providing power and the one where the electric line is active, either providing power or recharging batteries.

The states of the cooled subsystems were considered before because the TMs MSM diagram is a consequence of the thermal conditions of the cooled components, thus the diagram created for the TMS is shown in figure 6.9.

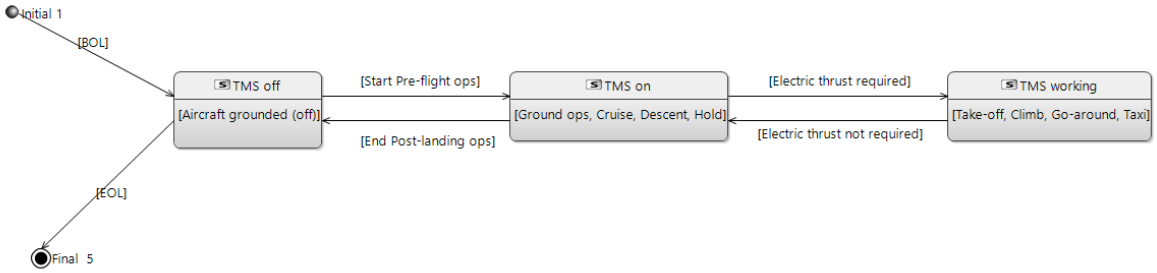

Figure 6.9: Thermal management system MSM diagram

Coherently with the data available at this stage of the process and with the MSM diagrams for the other subsystems, it is expected that the TMS would work during take-off, climb, go-around and taxi phases. The other flight phases require less cooling and so it is assumed for this system the existence of a "stand-by state", namely where the TMS is "on" but not at full capacity, this different cooling need is certainly a condition that could characterize phases where the electrical drive train is on but not providing full electrical power.

These models could be used as a reference where the engineers shall choose the heat loads and thus the condition that is going to be simulated. The goal of this part of the work is not to automate the heat load allocation, for instance linking an excel file with Capella, but to guide the engineers in the choice of the conditions to simulate.

Note that MSM diagrams are part of the so called "transverse modelling" in Capella, that means that such a type of diagram could be created at every Capella layer. The MSM diagrams previously shown were all built in the logical architecture layer, and they are thus connected with their respective logical components present in the architecture.

In the future it would certainly be interesting to add to this diagrams values that could describe each system state depicted, for example setting a heat load range for every state of every system. In this way there will be a set of diagrams that would make the engineers aware of the conditions that are going to be allocated in the model, inserting for instance the heat load values in the logical architecture diagram for the TMS there will be the possibility to check into the MSM diagrams in which range the selected value lies and which flight phases are connected to that state.

A possible development of this work was hypothesized to be the scenario description of each flight phase, as considered in section 5 it is not possible as the functional analysis did not bring to functions that specifically differentiates the behaviour along the mission. To do that there would be the need to explicit TMS and subsystem functions at logical level in the way that a temporal chain could be created in scenario diagrams, however now is available just the chain referring to the " general" cooling system process that takes place when heat removal is needed.

Having said that it makes more sense at this stage to create the framework considered in this section that refers to different system states but without having to detail them. In a future development either this framework will be enough for describing the systems states, allocating values, or more detailed functions will allow a "sequence" description of all flight phases in a set of scenario diagrams. In this last case it will might make sense to create the scenario description of the flight phases at a physical architecture level as at that level information about the component behaviour will allow to detail more both functions and requirements. Anyway, now that could be a future development since, as already said in the previous chapter, it does not make sense to consider the physical layer as there is not yet information that could be placed at that model level, further developments will allow to go in that system detail.

6.2.2 TMS physical architecture states and modes

The research about system states and modes brought to the conclusion that would be interesting to have the possibility to chose directly from the MBSE what flight phase to simulate and therefore on which conditions to size the TMS.

As previously said the main information that characterizes a flight phase is the heat load produced by the cooled subsystems, namely energy storage, power transmission and propulsive system. It makes sense for the work purpose to have some diagrams where there is the description of different heat loads for each component for each flight phase.

This means that an actual translation of the table in Appendix IV in the Capella environment needs to be made, that could be done through MSM diagrams.

The flight phases are not an internal state of the considered subsystems, they are more a set of imposed conditions determined by the mission profile that the aircraft is following, thus win the diagrams we are going to consider system "modes" instead of system "states".

To be coherent with the architecture developed in this physical layer MSM diagrams are created for "Batteries", "Distribution, inverters, converterts" and "eMotors and eBoosters" physical elements, these diagrams are shown in figures 6.10, 6.11, 6.12.

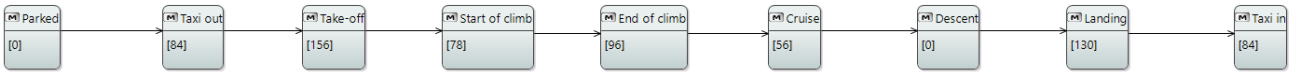

Figure 6.10: eMotors and eBoosters MSM diagram

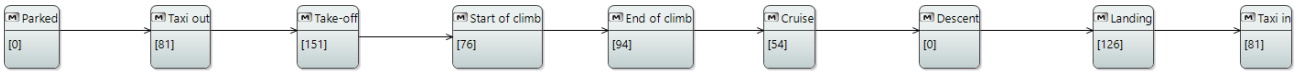

Figure 6.11: Batteries MSM diagram

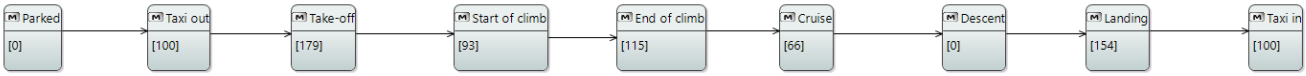

Figure 6.12: Distribution, inverters, converters MSM diagram

As can be observed the mode sequences for the three considered physical elements are always the same as they follow the same mission profile, what changes is the heat load, allocated in the region of each mode, that is produced by these components throughout the mission.

These diagrams are really the translation of the table in Appendix IV in the more readable MBSE environment, when more than one component is considered for a diagram, the respective heat loads in the Appendix IV are summed.

The reason to develop these diagrams in the physical architecture Capella layer is that they could not be just a reference to look at when performing the system sizing, indeed it will be interesting that the code previously built could access those data picking the right heat load values for the simulation.

In the physical architecture layer, more operations are allowed on the model with Python4Capella and indeed is possible to retrieve the information allocated in those diagrams, the following work will concern how to integrate this possibility exploiting the framework built in the physical layer, considered useless until this point, and the "modes" diagrams.

6.3 Link Matlab/Simulink models – Physical architecture

In this section it is explored the possibility to create another digital connection, like the one built in chapter 5, that could link the Matlab/Simulink models with the MBSE environment, this time it will be the Capella logical architecture layer.

Having already the model physical architecture, shown in paragraph 6.1, we want to create for the PAB diagram the same framework that was previously created for the LAB diagram. This means that we want to allocate all the values needed from the simulation and to have those properties values where all the simulations results will be stored at the end of the process.

This activity is really the same presented in paragraphs 5.2 and 5.3, now the main differences and peculiar aspects of this process in the physical layer will be shown.

The first thing to note is that the simulation models have not changed, thus there is no added value in the results respect to what made before, there are not simulation outcomes that could be addressed to one of the new created physical elements. Therefore, it is chosen to allocate all the results in the overall "TMS for electrical drive train" NodePC, this element is directly derived from the logical architecture and has the very same system high-level meaning, this element then contains the detailed TMS physical architecture but as said before it does not make sense to consider the results at component level.

The results considered are the same of the work done in section 5 and all the variable names have the same meaning, for this stage of the work the code was refined and thus now are added (figure 6.13):

- Run_result: output about the simulation run
- Run_phase: flight phase simulated.

(Physical Component) TMS for electrical drive train

Name	Value
COP_req	
Extension_1	
COP_req	0.0 %
Extension_2	
COP_req_bool	...
Mass	
Extension_1	
Mass	0.0 kg
Extension_2	
Mass_req	0.0 %
Extension_3	
Mass_req_bool	...
Power	
Extension_1	
Power_consumption	0.0 W
Extension_2	
Power_req	0.0 %
Extension_3	
Power_req_bool	...
Ram_air_consumption	
Extension_1	
Ram_air_consumption	0.0 kg/s
Run_result	
Extension_1	
Run_result	...
Extension_2	
Run_phase	...
SPD	
Extension_1	
SPD_req	0.0 %
Extension_2	
SPD_req_bool	...

Figure 6.13: TMS for electrical drive train data set (physical)

The information on the cooled components needed from the simulation are always the same, however this time it makes sense to allocate what possible into the new created physical elements as “Batteries”, “Distribution, inverters, converterts” and “eMotors and eBoosters”. This because the heat load values will be retrieved from the MSM diagrams associated with each of these specific physical elements, and so now there is no need to build a property for that value as done in the logical architecture. The other data needed are the cooling option that tells which model to run and the temperature range considered for a component, for simplicity in this stage they are created just a “Temperature” and a “Configuration” property values that will be allocate in each of those physical elements cooled.

(Physical Component) Batteries

Name	Value
Configuration	
Extension_1	
Configuration	0
Temperature	
Extension_1	
T	0.0 °C

Figure 6.14: Data set example for "Batteries", "Distribution, inverters, converters", "eMotors and eBoosters"

The last thing needed in this framework is the requirements allocation, also in this case the reasoning is the same of the one about the results allocation, since the requirements are not specific on some TMS subcomponents the most suitable thing is to allocate the requirements to the “TMS for electrical drive train” physical component that carries the same meaning that had in the logical architecture layer and that allows not to be to specify on a single component. Obviously, the requirements have not changed and are always the same previously presented.

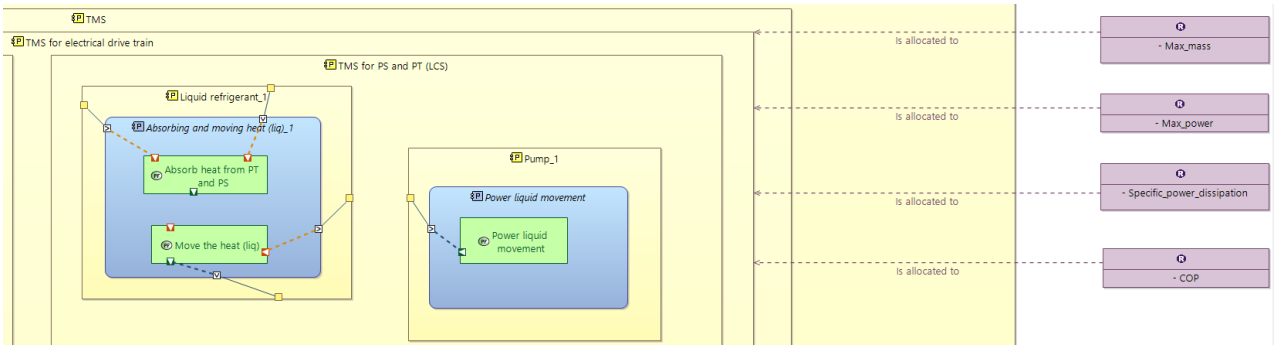

Figure 6.15: Requirements allocation in PAB diagram (zoom)

As soon as this framework is built we can create the script that performs the digital connection, this is very similar to the one created for the connection on the logical layer and significant parts of it are really the same.

What changes now is that the program gives the user the possibility to choose the flight phase to run, with a basic user interface the program allows the user to select the phase and then sends to the simulation the respective associated heat load value.

This is a significant improvement in the work as the new-created flight phases reference, that is the set of MSM diagrams with heat load allocation, is now used actively to initialise the sizing simulation. The values in the MSM diagrams could be updated basing on studies, research, and other evaluations but with the same code the user can always select what to run having a fast understanding of how

the TMS estimation changes depending on the flight phases and furthermore he is kept aware of the heat load associated with the simulated condition.

In this way MSM diagrams could result helpful on both the MBSE representation side, giving a faster readable way to look at heat loads in flight phases for each cooled subsystems, and the digital connection side as the simulation dynamically picks the values associated with the phases that the user, namely an engineer, is choosing to run. Furthermore the phase simulated is placed as a result in the "Phase" property value allocated in the "TMS for electrical drive train" element.

A further improvement, that does not concern the simulation itself, is that now is given the possibility to choose the output that is printed in the console, anyway all the results are always allocated in the model as shown in chapter 5. Now the user can choose to have either all results printed in the console, or to have just verification results, namely the ones that say whether the requirements are satisfied or not, or not to have results printed in the console.

In any case the results are placed in the model and the summary about single subsystems is always printed out while running.

Furthermore, an update in the P4C API allows now to modify from the code some requirement attributes, to make the requirements results as visible as possible they are also placed in the "Prefix" attribute of the requirement elements.

The Matlab/Simulink models are built to manage in the best way those conditions that could be meaningful for the sizing evaluation, for instance conditions with a significant heat load. Now, since the simulation can receive as input low heat load values associated to phases like "Cruise" or phases where that value is zero like in "Descent" and "Cruise", there is the need to also manage those results with the code.

The issue is that the Simulink model cannot run and gives an error when the heat load value is zero, when the heat load is low another problem persist as the simulation could give, depending also on the temperature value and the configuration option, a negative power consumption and a negative ram air consumption. These results do not have a real physical meaning but are representative of a condition where cooling is not needed for that element, this is an assumption, but it could not affect that much the simulation results as such a thing happens when the heat load value is an order on magnitude lower than the value considered in those conditions supposed as sizing phases.

Given this assumption the code manages these situations considering not necessary the TMS evaluation when the heat load is zero, thus not running the simulation. In the case of low heat load the code analyses the results and in the case of negative power consumption it supposes that a TMS is not necessary in that condition and imposes that for that evaluation, that could be for instance just for one of the cooled elements, that mass, power consumption and ram air consumption are all equal to zero, therefore a non-existent TMS.

At the end of all three simulations, one for each cooled element, if the overall result is equal to zero then the TMS is supposed not to be necessary in that specific case and so all results are imposed as zero, the requirement comparison is not performed and at the end in the "Run_result" property value it would be printed "TMS not evaluated". In all the other cases the results are compared with the requirements and all the results end in the model as it was in the logical architecture, furthermore in "Run_result" property is written now "TMS evaluated".,

The code now can manage all the conditions available in the MSM diagrams, anyway it must be considered that are the engineers that can choose what values to place in the diagram and what phase to run, and since the engineer will search for demanding and sizing conditions it would not be that interested in situation where there is no heat waste produced. However now the code would not give an error when in those cases and will allow the engineers to follow their work evaluating different conditions, and in the case that the data in the MSM diagrams would be updated the program will always be able to give useful results to the user.

For example, if from other analysis it will be supposed that during "Descent" there will be needed some cooling in certain components, a value different from zero will be allocated there and there will be immediately the possibility to simulate it without changing anything in the program.

This framework allows the engineers to simulate several conditions, namely all the phases of the mission profile, in a very fast way and having immediately meaningful results in terms also of requirements comparison. If the reference mission changes, or changes the behaviour of the components throughout it, the heat load values could be manually updated in the MSM diagrams and then there will be the possibility to check immediately the system sizing along the different phases of the new mission.

The following paragraph will show how this new digital "bridge" can be used and especially what are the differences with the link created in chapter 5.

6.4 Phase two results

In this paragraph is reported a sort of tutorial that is ideally the extension of what explained in section 5.4. Now there is no more the need to show a numerical example or all the process that must be followed to use the digital tool, instead since the program was refined here are reported the differences and the significant improvements of the digital link.

As previously said the new digital connection created for the physical architecture does not change the simulation models and therefore the results interpretation is actually the same, also the results allocation is made in the way to be as coherent as possible with the logical layer, this because the meaning of the simulation outcome is more related to the "TMS for electrical drive train" logical element rather then any subcomponent considered in the physical architecture.

Anyway a thing that we must note is that the TMS configuration option, that could be addressed to the three cooled subsystem is not a variable anymore. That property value describes which cooling option will be implemented but now in the physical architecture there is the actual representation of that option.

For instance, if we consider the "Transition 1" physical model (figure 6.3), the architecture is relative to a TMS layout where batteries are cooled with an option 1 configuration and all the other components are cooled with option 3. Thus, even though is possible to select another type of cooling option and to simulate it without having either errors or simulation issues, it does not really make sense to evaluate them, that would result in having simulation evaluation that does not correspond to the built architecture. That could certainly confuse and give wrong insights on the TMS sizing and thus the engineers must be aware of that.

Apart from that there are no other differences with what reported in chapter 5 about results.

Even though being developed for the physical architecture layer the interaction between this tool and the engineering workflow does not change, it always aims to be a sharable tool that boosts the information exchange between engineers.

Now the process described in section 5.4 is really the same, but it is based in the physical architecture, now the values to allocate in the cooled components are just the configuration, that will be fixed as previously explained, and the reference temperature for the physical element considered.

There is no need to place the heat load as it is already present in the phase description for the cooled elements in those MSM diagrams.

Then the user can immediately run the "digital connection tool" and what is interesting to show now is the interfaces that the program now creates to allow the user to interact with the simulation.

First the user can select the output in the console writing the relative number in the dedicated space (figure 6.16), the program does not keep running until one of the options is selected.

Then the user can select which flight phase to run always inserting the number associated with the desired phase (figure 6.17)

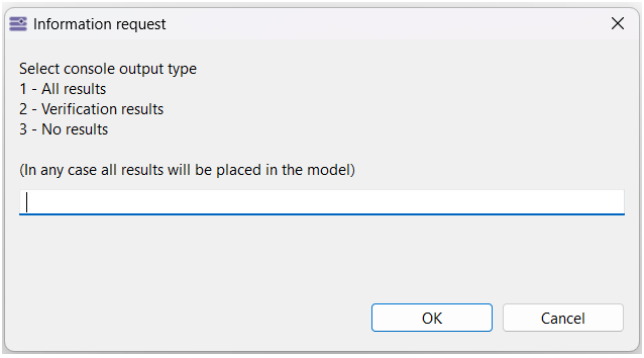

Figure 6.16: User console results choice

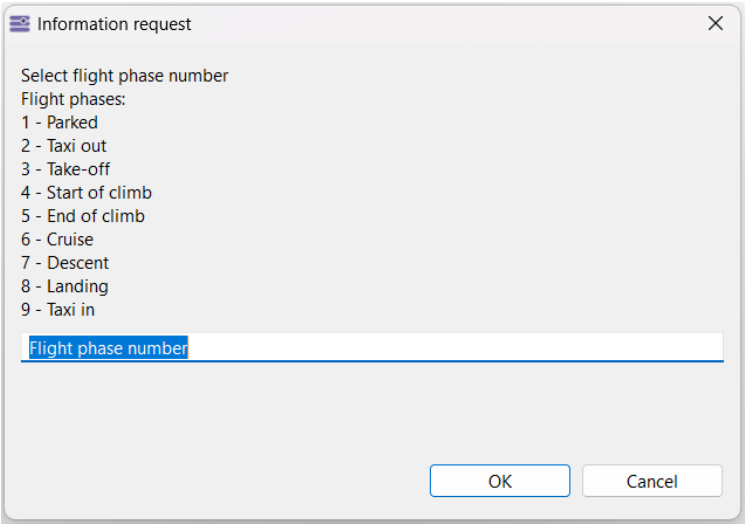

Figure 6.17: User flight phase choice

At this point in the console is printed for instance:

```
TMS sizing on "Parked" conditions
```

Then the program works in the same way as shown in section 5.4, thus running for instance a phase where the heat load is significant, that means excluding just “Parked”, “Descent”, and “Cruise”, there are no changes in the program behaviour.

However, is interesting to show the result section printed in the console when the option “Verification result” is chosen, in that case an example of output would be:

```
---------RESULTS--------

Mass REQUIREMENT SATISFIED
Power REQUIREMENT SATISFIED
Specific power dissipation REQUIREMENT NOT SATISFIED
COP REQUIREMENT SATISFIED
```

Note that before the result section in the console are always printed the results of the single component simulations.

Anyway, with this output the user has an immediate understanding of what is satisfied or not without having to check all the numbers.

In the case of “All results” the output is the same shown is section 5.4, while in case of “No results” in the console is printed just: “All results allocated in the model”.

Speaking about other differences, as said in the previous paragraph, there is now the possibility that for some of the cooled components the TMS could not be evaluated if the heat load is either to low or zero. In that case instead of the results shown for that specific element is printed for instance:

```
Power transmission
  Configuration 2
  N° 1
  Heat load 50.0 KW
  T max 393.0 K°
  T min 353.0 K°
  Cooling not required for Power transmission in Cruise phase
```

That could happen for more then one component and if it happens for all considered cooled elements the TMS results not to be necessary, for example running the “Parked” phase where the heat load is zero for all subsystems, the output in the console in the results section is:

```
Parked does not require cooling, TMS not evaluated
```

And in the results property values it is stated “TMS not evaluated”, in all the other cases there will always be at least one result and so those results will be placed in the model with the statement “TMS evaluated”.

An example is shown in figure 6.18 where the simulation results are allocated in the property values associated with the “TMS for electrical drive train”.

(Physical Component) TMS for electrical drive train

Name	Value
COP_req	
Extension_1	
COP_req	172.16 %
Extension_2	
COP_req_bool	True
Mass	
Extension_1	
Mass	1030.0 kg
Extension_2	
Mass_req	68.67 %
Extension_3	
Mass_req_bool	True
Power	
Extension_1	
Power_consumption	57387.47 W
Extension_2	
Power_req	75.51 %
Extension_3	
Power_req_bool	True
Ram_air_consumption	
Extension_1	
Ram_air_consumption	4.72 kg/s
Run_result	
Extension_1	
Run_result	TMS evaluated
Extension_2	
Run_phase	Start of climb
SPD	
Extension_1	
SPD_req	47.96 %
Extension_2	
SPD_req_bool	False

Figure 6.18: Result allocation in the property values set of TMS for electrical drive train NodePC

In the case that the TMS is not evaluated all result values are imposed as either zero, for numerical values, or "…", for Boolean and string values.

As previously said the last updated of this refine code is that the requirements results are also showing the "Prefix" field that is visible clicking on the requirements blocks in the diagram:

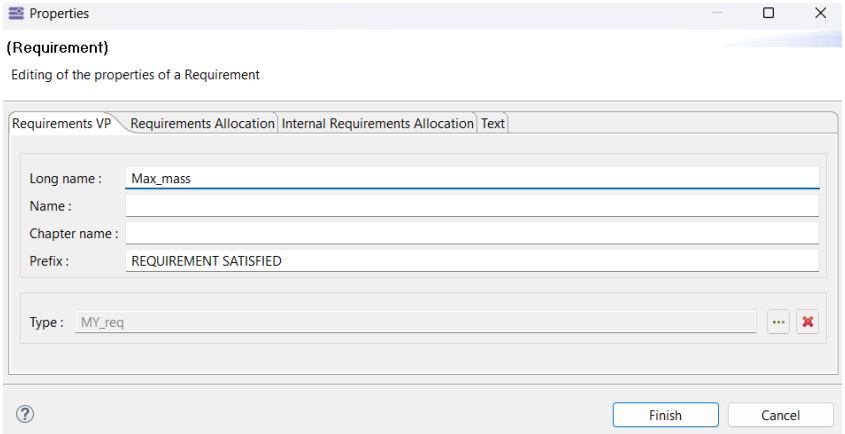

Figure 6.19: Example of verification result placed in the "Prefix" field in the requirement block

The script the performs this new digital connection on the physical architecture Capella layer is reported in Appendix VI, as explained some sections of it are the same of the previous script but the

changes implemented have allowed this more refined digital bridge with the user interaction capability.

With this section the work on the "digital tool" is concluded, the refining work allowed by the Capella features available in the physical architecture layer, brought to this last version of the connection. Now the built framework is representative of the system that is being built and allows the allocation of several information and values inside it, also in the future the physical architecture could allow a further description of the system. The framework relates to the sizing Matlab and Simulink models whose results are compared with requirements and placed in the model, for this purpose the physical architecture is not the most suitable layer where to place these results, but its features are exploited to provide that consistent connection between the architecture and the flight phase description for the cooled components. This in the end allows the manipulation of all those data to give the user the possibility to dynamically select which phase to run and significantly boosting the engineering process.

In terms of model meaning it will make more sense in the future to built this kind of connection in the physical layer but with dynamic simulation models that will allow to verify the component behaviour and thus allow the comparison with requirements at component level when they will be available. Now the result meaning is still on a higher "logical" level while the framework exploited is the physical one. This issue will be solved in the future with further developments on the system conceptual design side and on the simulation development side.

7 Conclusions

The development of an MBSE model in Capella for a thermal management system (TMS) for a hybrid-electric aircraft, and the link of this model with the sizing models developed in the Matlab/Simulink environment, is presented in this paper. The research was conducted in collaboration with Embraer Research and Technology Europe - Airholding S.A. company. The objective of this research was to develop a digital tool that enables a faster and more effective information exchange between all the engineering teams involved in the activities.

The reference material provided by the company, including the aircraft, the thermal management system with some possible layouts, and respective available requirements for aircraft and TMS, was analysed in detail. Also, the sizing Matlab/Simulink models provided by the company were analysed to understand the engineering workflow that is going on to develop the TMS and the possible MBSE practices that could boost the process.

The scope of this project was identified, which is the digital tool formed by an MBSE model linked with simulation models in Matlab/Simulink. The first phase of the project involved the development of the MBSE architecture for the TMS in Capella keeping an aircraft perspective, allocation of all requirements and values to characterize the model, and the development of the digital link between the Capella logical architecture layer and Matlab/Simulink model through Python4Capella. The simulation results and the results of the first comparison with requirements were made available in the model.

The second phase regarded investigating the possible utility of developing the more detailed Capella physical architecture layer and trying to characterize the model with the description of cooled subsystems states and modes throughout the mission. The physical architecture layer and the modes diagrams for the cooled subsystems were exploited to create a refined digital connection that now uses the MSM diagrams as a reference. This allows the user to choose which flight phase to simulate, and the program can dynamically manage the simulation for all flight phases. This refined connection links now the Matlab/Simulink models with the physical architecture layer where now results are placed.

The created framework and digital link in the end could boost the system development process, allowing a faster and more effective information exchange between all the engineering teams involved in the activities. The model is consistent with the material provided by the company and eventually allows also to bring a similar approach for the other subsystems.

The work presented in this paper gives immediately helpful insights on the ongoing project and satisfies the initial goal and the needs that motivated it. The results of this research provide a proof-of-concept for the development of an MBSE model in Capella for a thermal management system for a hybrid-electric aircraft. The use of MBSE practices in the development process has been shown to be effective in providing a digital tool that can significantly reduce the time associated with the design and development of the TMS.

Future work can involve further development of the MBSE model to incorporate more subsystems and expand the scope of the digital tool. Moreover, a more detailed analysis of the use of MBSE practices in the development process can be carried out, to identify potential areas for further improvement. The proposed digital tool can be further refined and customized to meet specific

project requirements. The presented research can serve as a starting point for further research and development of MBSE models for complex systems such as the TMS for hybrid-electric aircraft.

In conclusion, this paper provides a comprehensive account of the development of an MBSE model in Capella for a thermal management system for a hybrid-electric aircraft. The results of this research demonstrate the effectiveness of the proposed digital tool in improving the engineering workflow, reducing time and eventually costs, and providing a faster and more effective information exchange between all the engineering teams involved in the activities of developing the TMS starting from a conceptual outline of the system.

The project result is coherent with what demanded from the work motivation, the solution proposed goes in the direction of solving the document-based engineering issues and enhancing the MBSE capabilities through the connection with Matlab/Simulink. The results improve our knowledge as now the single point of truth reference that the MBSE approach aims to create has branched in a new way towards the numerical environment, on the other hand this latter environment uses MBSE a communication mean to reach in a more consistent and effective way all the stakeholders involved in the design process.

8 Future developments

In this chapter are mentioned those topics that were considered at the end of the work but that eventually had not been part of the main result of it.

To make the model more understandable and the requirements comparison outcome immediately visible in the diagram where requirements are placed, namely the PAB diagram. An attempt was made to automatically highlight and differentiate those requirement element blocks basing on the whether the requirement was satisfied or not. This attempt regarded the research to make this model modification directly through the P4C code so that the information available from the requirement comparison, performed by the program, could be immediately exploited.

Unfortunately, the Capella software and some of its features and add-ons are still under development, thus some operations were not available on the model especially with Python4Capella, the simplified API indeed presents some incomplete functions and methods that had not allowed to either change the colour or hide the elements in the diagrams.

In conclusion it was not possible to create a visible distinction, in terms of element format, between requirements satisfied and not satisfied, either hiding the fulfilled requirements or changing their colours. Further updates on the software and on the add-on could in the future allow this kind of operation on the model diagrams.

The last unclosed topic was about the type of connection that occurs between Matlab/Simulink with the logical architecture and Matlab/Simulink with the physical architecture. It was already explained how the physical architecture layer features were exploited to make the digital link refinement, but that the results of the simulation make more sense on the logical level of the conceptual system development. However, in the future, if further updates to the environment will allow it, it will be interesting to improve the digital link architecture sticking on the logical layer for these kind of sizing evaluations. The physical architecture could be exploited in a similar way but connecting a dynamic simulation model, that would verify the behaviour of the system in the mission time span, instead of the sizing model.

This section concludes the work as no other elements were considered to complete the project, it is hoped these last considerations could be an interesting cue for future developments of this kind of framework in the MBS environment.

Appendix I

- Air thermophysical properties (ISA)

Temperature [°C]	Specific heat [J/(kg K)]	Viscosity [Pa s]	Kinematic viscosity [(J s)/kg]	Thermal conductivity [W/(m K)]	Thermal diffusivity [m^2/s]	Pr
100	1.032	71.1	2.00	9.34	2.54	0.786
150	1.012	103.4	4.426	13.8	5.84	0.758
200	1.007	132.5	7.590	18.1	10.3	0.737
250	1.006	159.6	11.44	22.3	15.9	0.720
300	1.007	184.6	15.89	26.3	22.5	0.707
350	1.009	208.2	20.92	30.0	29.9	0.700
400	1.014	230.1	26.41	33.8	38.3	0.690
450	1.021	250.7	32.39	37.3	47.2	0.686
500	1.030	270.1	38.79	40.7	56.7	0.684
550	1.040	288.4	45.57	43.9	66.7	0.683
600	1.051	305.8	52.69	46.9	76.9	0.685
650	1.063	322.5	60.21	49.7	87.3	0.690
700	1.075	338.8	68.10	52.4	98.0	0.695
750	1.087	354.6	76.37	54.9	109	0.702
800	1.099	369.8	84.93	57.3	120	0.709
850	1.110	384.3	93.80	59.6	131	0.716
900	1.121	398.1	102.9	62.0	143	0.720
950	1.131	411.3	112.2	64.3	155	0.723
1000	1.141	424.4	121.9	66.7	168	0.726
1100	1.159	449.0	141.8	71.5	195	0.728
1200	1.175	473.0	162.9	76.3	224	0.728
1300	1.189	496.0	185.1	82	257	0.719
1400	1.207	530	213	91	303	0.703
1500	1.230	557	240	100	350	0.685
1600	1.248	584	268	106	390	0.688
1700	1.267	611	298	113	435	0.685
1800	1.286	637	329	120	482	0.683
1900	1.307	663	362	128	534	0.677
2000	1.337	689	396	137	589	0.672
2100	1.372	715	431	147	646	0.667
2200	1.417	740	468	160	714	0.655
2300	1.478	766	506	175	783	0.647
2400	1.558	792	547	196	869	0.630
2500	1.665	818	589	222	960	0.613
3000	2.726	955	841	486	1570	0.536

- Ethylene-Glycol properties

Temperature [°C]	Specific heat [J/(kg K)]	Viscosity [Pa s]	Thermal conductivity [W/(m K)]
238,2	2,844	0,09344	0,307
243,2	2,866	0,06525	0,312
248,2	2,888	0,04675	0,316
253,2	2,909	0,03428	0,321
258,1	2,931	0,02569	0,325
263,1	2,953	0,01962	0,329
268,1	2,975	0,01525	0,333
273,1	2,997	0,01205	0,336
278,1	3,018	0,00966	0,34
283,1	3,04	0,00785	0,343
288,1	3,062	0,00646	0,346
293,1	3,084	0,00538	0,349
298,1	3,106	0,00452	0,352
303,1	3,127	0,00384	0,355
308,1	3,149	0,00329	0,358
313,1	3,171	0,00284	0,36
318,1	3,193	0,00247	0,363
323,1	3,215	0,00216	0,365
328,1	3,236	0,00191	0,367
333,1	3,258	0,00169	0,369
338,1	3,28	0,00151	0,371
343,1	3,302	0,00135	0,372
348,1	3,324	0,00122	0,374
353,1	3,345	0,0011	0,375
358,1	3,367	0,001	0,376
363,1	3,389	0,00092	0,377
368,1	3,411	0,00084	0,378
373,1	3,433	0,00077	0,379
378,1	3,454	0,00071	0,379
383,1	3,476	0,00066	0,38
388,1	3,498	0,00061	0,38
393,1	3,52	0,00057	0,38
398,1	3,542	0,00053	0,38

- Satured R314 properties

Temperature [°C]	ressure [ba	Specific volume f [m^3/kg]	Specific volum g [m^3/kg]	Internal energy f [KJ/kg]	Internal energy g [KJ/kg]	Enthalpy f [KJ/kg]	Enthalphy gf [KJ/kg]	Enthalpy g [KJ/kg]	Entropy f [KJ/(kg K)]	Entropy g [KJ/(kg K)]
-40	0,5164	0,7055	0,3569	-0,04	204,45	0	222,88	222,88	0	0,956
-36	0,6332	0,7113	0,2947	4,68	206,73	4,73	220,67	225,4	0,0201	0,9506
-32	0,7704	0,7172	0,2451	9,47	209,01	9,52	218,37	227,9	0,0401	0,9456
-28	0,9305	0,7233	0,2052	14,31	211,29	14,37	216,01	230,38	0,06	0,9411
-26	1,0199	0,7265	0,1882	16,75	212,43	16,82	214,8	231,62	0,0699	0,939
-24	1,116	0,7296	0,1728	19,21	213,57	19,29	213,57	232,85	0,0798	0,937
-22	1,2192	0,7328	0,159	21,68	214,7	21,77	212,32	234,08	0,0897	0,9351
-20	1,3299	0,7361	0,1464	24,17	215,84	24,26	211,05	235,31	0,0996	0,9332
-18	1,4483	0,7395	0,135	26,67	216,97	26,77	209,76	236,53	0,1094	0,9315
-16	1,5748	0,7428	0,1247	29,18	218,1	29,3	208,45	237,74	0,1192	0,9298
-12	1,854	0,7498	0,1068	34,25	220,36	34,39	205,77	240,15	0,1388	0,9267
-8	2,1704	0,7569	0,0919	39,38	222,6	39,54	203	242,54	0,1583	0,9239
-4	2,5274	0,7644	0,0794	44,56	224,84	44,75	200,15	244,9	0,1777	0,9213
0	2,9282	0,7721	0,0689	49,79	227,06	50,02	197,21	247,23	0,197	0,919
4	3,3765	0,7801	0,06	55,08	229,27	55,35	194,19	249,53	0,2162	0,9169
8	3,8756	0,7884	0,0525	60,43	231,46	60,73	191,07	251,8	0,2354	0,915
12	4,4294	0,7971	0,046	65,83	233,63	66,18	187,85	254,03	0,2545	0,9132
16	5,0416	0,8062	0,0405	71,29	235,78	71,69	184,52	256,22	0,2735	0,9116
20	5,716	0,8157	0,0358	76,8	237,91	77,26	181,09	258,36	0,2924	0,9102
24	6,4566	0,8257	0,0317	82,37	240,01	82,9	177,55	260,45	0,3113	0,9089
26	6,853	0,8309	0,0298	85,18	241,05	85,75	175,73	261,48	0,3208	0,9082
28	7,2675	0,8362	0,0281	88	242,08	88,61	173,89	262,5	0,3302	0,9076
30	7,7006	0,8417	0,0265	90,84	243,1	91,49	172	263,5	0,3396	0,907
32	8,1528	0,8473	0,025	93,7	244,12	94,39	170,09	264,48	0,349	0,9064
34	8,6247	0,853	0,0236	96,58	245,12	97,31	168,14	265,45	0,3584	0,9058
36	9,1168	0,859	0,0223	99,47	246,11	100,25	166,15	266,4	0,3678	0,9053
38	9,6298	0,8651	0,021	102,38	247,09	103,21	164,12	267,33	0,3772	0,9047
40	10,164	0,8714	0,0199	105,3	248,06	106,19	162,05	268,24	0,3866	0,9041
42	10,72	0,878	0,0188	108,25	249,02	109,19	159,94	269,14	0,396	0,9035
44	11,299	0,8847	0,0177	111,22	249,96	112,22	157,79	270,01	0,4054	0,903
48	12,526	0,8989	0,0159	117,22	251,79	118,35	153,33	271,68	0,4243	0,9017
52	13,851	0,9142	0,0142	123,31	253,55	124,58	148,66	273,24	0,4432	0,9004
56	15,278	0,9308	0,0127	129,51	255,23	130,93	143,75	274,68	0,4622	0,899
60	16,813	0,9488	0,0114	135,82	256,81	137,42	138,57	275,99	0,4814	0,8973
70	21,162	1,0027	0,0086	152,22	260,15	154,34	124,08	278,43	0,5302	0,8918
80	26,324	1,0766	0,0064	169,88	262,14	172,71	106,41	279,12	0,5814	0,8827
90	32,435	1,1949	0,0046	189,82	261,34	193,69	82,63	276,32	0,638	0,8655
100	39,742	1,5443	0,0027	218,6	248,49	224,74	34,4	259,13	0,7196	0,8117

- Superheated R314 properties (pressure=14 bar)

Temperature [°C]	Specific volume [m^3/kg]	Internal energy [KJ/kg]	Enthalpy [KJ/kg]	Entropy [KJ/(kg K)]
52,43	0,01405	253,74	273,4	0,9003
60	0,01495	262,17	283,1	0,9297
70	0,01603	272,87	295,31	0,9658
80	0,01701	283,29	307,1	0,9997
90	0,01792	293,55	318,63	1,0319
100	0,01878	303,73	330,02	1,0628
110	0,0196	313,88	341,32	1,0927
120	0,02039	324,05	352,59	1,1218
130	0,02115	334,25	363,86	1,1501
140	0,02189	344,5	375,15	1,1777
150	0,02262	354,82	386,49	1,2048
160	0,02333	365,22	397,89	1,2315
170	0,02403	375,71	409,36	1,2576
180	0,02472	386,29	420,9	1,2834
190	0,02541	396,96	432,53	1,3088
200	0,02608	407,73	444,24	1,3338

- Superheated R314 properties (pressure=12 bar)

Temperature [°C]	Specific volume [m^3/kg]	Internal energy [KJ/kg]	Enthalpy [KJ/kg]	Entropy [KJ/(kg K)]
46,32	0,01663	251,03	270,99	0,9023
50	0,01712	254,98	275,52	0,9164
60	0,01835	265,42	287,44	0,9527
70	0,01947	275,59	298,96	0,9868
80	0,02051	285,62	310,24	1,0192
90	0,0215	295,59	321,39	1,0503
100	0,02244	305,54	332,47	1,0804
110	0,02335	315,5	343,52	1,1096
120	0,02423	325,51	354,58	1,1381
130	0,02508	335,58	365,68	1,166
140	0,02592	345,73	376,83	1,1933
150	0,02674	355,95	388,04	1,2201
160	0,02754	366,27	399,33	1,2465
170	0,02834	376,69	410,7	1,2724
180	0,02912	387,21	422,16	1,298

- Superheated R314 properties (pressure=4 bar)

Temperature [°C]	Specific volume [m^3/kg]	Internal energy [KJ/kg]	Enthalpy [KJ/kg]	Entropy [KJ/(kg K)]
8,93	0,05089	231,97	252,32	0,9145
10	0,05119	232,87	253,35	0,9182
20	0,05397	241,37	262,96	0,9515
30	0,05662	249,89	272,54	0,9837
40	0,05917	258,47	282,14	1,0148
50	0,06164	267,13	291,79	1,0452
60	0,06405	275,89	301,51	1,0748
70	0,06641	284,75	311,32	1,1038
80	0,06873	293,73	321,23	1,1322
90	0,07102	302,84	331,25	1,1602
100	0,07327	312,07	341,38	1,1878
110	0,0755	321,44	351,64	1,2149
120	0,07771	330,94	362,03	1,2417
130	0,07991	340,58	372,54	1,2681
140	0,08208	350,35	383,18	1,2941

- Superheated R314 properties (pressure=3.2 bar)

Temperature [°C]	Specific volume [m^3/kg]	Internal energy [KJ/kg]	Enthalpy [KJ/kg]	Entropy [KJ/(kg K)]
2,48	0,06322	228,43	248,66	0,9177
10	0,06576	234,61	255,65	0,9427
20	0,06901	242,87	264,95	0,9749
30	0,07214	251,19	274,28	1,0062
40	0,07518	259,61	283,67	1,0367
50	0,07815	268,14	293,15	1,0665
60	0,08106	276,79	302,72	1,0957
70	0,08392	285,56	312,41	1,1243
80	0,08674	294,46	322,22	1,1525
90	0,08953	303,5	332,15	1,1802
100	0,09229	312,68	342,21	1,2076
110	0,09503	322	352,4	1,2345
120	0,09774	331,45	362,73	1,2611

Appendix II

- Example of pre-processing Matlab script

```
%Table data inputs
% Air properties (ISA)

thermophyscialpropertiesair2 = readtable('../Thermophysicalproperties_air.txt');
thermophyscialpropertiesair2.Properties.VariableNames =
{'Temperature','Specificheat','Viscosity','Kinematicviscosity','Thermalconductivity','Thermal
diffusivity','Pr'};
T_air=thermophyscialpropertiesair2.Temperature;
cp_air=thermophyscialpropertiesair2.Specificheat;
Mi_air=thermophyscialpropertiesair2.Viscosity;
Pr_air=thermophyscialpropertiesair2.Pr;



% Thermal liquid properties (ehylene-glycol)

thermophyscialpropertiesethywater = readtable('..\EGW60_40properties.xlsx');
thermophyscialpropertiesethywater.Properties.VariableNames =
{'Temperature','Specificheat','Viscosity','Thermalconductivity'};
T_ethywat=thermophyscialpropertiesethywater.Temperature;
cp_ethywat=thermophyscialpropertiesethywater.Specificheat;
Mi_ethywat=thermophyscialpropertiesethywater.Viscosity;
k_ethywat=thermophyscialpropertiesethywater.Thermalconductivity;


%Satured R314 properties
thermopropertiesR314a = readtable('..\R314_Temperaturetable.xlsx');
'Pressure','Specificvolumef','Specificvolumeg','InternalEnergyf','InternalEnergyg','Enthalpyf
','Enthalpygf','Enthalpyg','Entropyf','Entropyg','Temp2'};
T_R314a=thermopropertiesR314a.Temperature;
p_R314a=thermopropertiesR314a.Pressure;
vf_R314a=thermopropertiesR314a.Specificvolumef;
vg_R314a=thermopropertiesR314a.Specificvolumeg;
uf_R314a=thermopropertiesR314a.InternalEnergyf;
ug_R314a=thermopropertiesR314a.InternalEnergyg;
hf_R314a=thermopropertiesR314a.Enthalpyf;
hgf_R314a=thermopropertiesR314a.Enthalpygf;
hg_R314a=thermopropertiesR314a.Enthalpyg;
sf_R314a=thermopropertiesR314a.Entropyf;
sg_R314a=thermopropertiesR314a.Entropyg;



%Superheated R314 properties p=12bar

thermopropertiesR314ah_12 = readtable('..\SuperheateadR314a_12.xlsx');
thermopropertiesR314ah_12.Properties.VariableNames = {'Temperature',
'Specificvolume','InternalEnergy','Enthalpy','Entropy'};
T_R314ah_12=thermopropertiesR314ah_12.Temperature;
v_R314ah_12=thermopropertiesR314ah_12.Specificvolume;
u_R314ah_12=thermopropertiesR314ah_12.InternalEnergy;
h_R314ah_12=thermopropertiesR314ah_12.Enthalpy;
s_R314ah_12=thermopropertiesR314ah_12.Entropy;



%Superheated R314 properties p=14bar

thermopropertiesR314ah_14 = readtable('..\SuperheatedR314a_14.xlsx');
thermopropertiesR314ah_14.Properties.VariableNames = {'Temperature',
'Specificvolume','InternalEnergy','Enthalpy','Entropy'};
```

```
T_R314ah_14=thermopropertiesR314ah_14.Temperature;
v_R314ah_14=thermopropertiesR314ah_14.Specificvolume;
u_R314ah_14=thermopropertiesR314ah_14.InternalEnergy;
h_R314ah_14=thermopropertiesR314ah_14.Enthalpy;
s_R314ah_14=thermopropertiesR314ah_14.Entropy;

%Superheated R314 properties p=3.2bar

thermopropertiesR314ah3_2 = readtable('..\SuperheateadR314a_3_2.xlsx');
thermopropertiesR314ah3_2.Properties.VariableNames = {'Temperature',
'Specificvolume','InternalEnergy','Enthalpy','Entropy'};
T_R314ah3_2=thermopropertiesR314ah3_2.Temperature;
v_R314ah3_2=thermopropertiesR314ah3_2.Specificvolume;
u_R314ah3_2=thermopropertiesR314ah3_2.InternalEnergy;
h_R314ah3_2=thermopropertiesR314ah3_2.Enthalpy;
s_R314ah3_2=thermopropertiesR314ah3_2.Entropy;


%Superheated R314 properties p=4bar

thermopropertiesR314ah_4 = readtable('..\SuperheateadR314a_4.xlsx');
thermopropertiesR314ah_4.Properties.VariableNames = {'Temperature',
'Specificvolume','InternalEnergy','Enthalpy','Entropy'};
T_R314ah_4=thermopropertiesR314ah_4.Temperature;
v_R314ah_4=thermopropertiesR314ah_4.Specificvolume;
u_R314ah_4=thermopropertiesR314ah_4.InternalEnergy;
h_R314ah_4=thermopropertiesR314ah_4.Enthalpy;
s_R314ah_4=thermopropertiesR314ah_4.Entropy;

%Input

npoints=60; %Number of saing point during simulation
T_InEquip=298; %T_equp required [K] (battery at 20 °C, controllers ESC converters inverters
(power electronics) at 60 °C, motors at (115°C)
Q_equip=500; %Heat load [kw]
mdot_liquid=0.045*Q_equip; %Liquid mass flow [kg/s] (between 0.023*Q_equip and 0.045*Q_equip
regulation)- regulate to an output of about 6 or 7 kg/s (+-)
mdot_ref=0.9; %Refrigerant mass flow [kg/s]
bypass=0; %By-pass ratio
T_evap=278; %Evaporation Temperature (K) - 20°C de diferença
T_cond=325.3; %Condensation Temperature (K)

% prompt = input(['Enter 1 for reference case or ' ...
%     '2 for varying the heat load:']); % ask
%
% choise = prompt;
%     if choise==1
%         Q_equip=100;
%     elseif choise==2
%         Q_equip=[100 150 200 250];
%     end


%Flight Conditions

decol=120:60:900;
cruise=(900+1000/3):(1000/3):1900;
land=(1900+1700/9):(1700/9):3601;
Time2=[0 decol cruise land];

R=287; %Universal gas constant (J/kgK)
Mach=[0.2774 0.2798 0.2823 0.3046 0.3074 0.3102 0.3345 0.3376 0.3408 0.3671 0.3707 0.3744
0.3913 0.3936 0.396 0.4613 0.4613 0.4613 0.4119 0.4053 0.3991 0.3512 0.3492 0.3472 0.3156
0.3088 0.3024]; %Flight Mach variation
gamma=1.4;
H=[0 762 1524 1524 2286 3048 3048 3810 4572 4572 5334 6096 6096 6553 7010 7010 7010 7010 7010
5791 4572 4572 4115 3658 3658 1829 0]; %Flight height variation (m)
```

```
[T_env, a_env, p_env, rho_env] = atmosisa(H); %Standard Atmosphere
fp=length(H); %Number of flight points considerer

%Geometric parameters

L_duct_liquid=20; %Liquid Duct Length [m]
t_duct_liquid = 0.0025; %Duct thickness [m]
L_duct_vcs = 10; %Duct length [m]
duct_roughness = 0.0025*0.001; %Duct roughness
D_vcs = 0.0254; %VCS duct diameter [m]
t_duct_vcs = 0.00117; %VCS duct thickness [m]
h = 0.01; %Thermal boundary layer thickness [m]
SHX_A=5; % SHX Area [m^2]
A_cond = 2; %Condenser area [m^2]
U_cond = 50; %Condenser heat transfer coeffient [W/m^2K]


%Pressure losses

DELTAP_evaporator = 5; %Pressure loss of evaporator [kPa]
DELTAP_condenser = 1.013; %Pressure loss of condenser [kPa]
DELTAP_equip = 5; %Pressure loss of heat load [kPa]
N_90dgBend = 4; %Pressure loss on bends
K_90dgBend = 0.83;
N_expansion = 0;
K_expansion = 0.3;
N_contraction = 0;
K_contraction = 0.03;
N_branch_converg = 0;
K_branch_converg = 0.05;
N_branch_diverg = 0;
K_branch_diverg = 0.05;


%VCS Cycle Parameters
DELTAT_subcool = 5; %[K]
DELTAT_superheat = 5; %[K]

%Efficiencies
eta_isentropic_compression = 0.8;%Value based on results of BE3888 Issue 2 of Secan
eta_compressor = 0.8; %Compression efficiency
eta_pump = 0.5;       %Pump efficiency
eta_fan = 0.4;%Fan efficiency
compress_ratio = 4;   %Compressor ratio;
```

- Example of post-processing Matlab script

```
%% RESULTS
%Liquid cooling
fp=npoints+1;
Q_evap=repelem(out.simoutevap,fp)'; %Heat load in evaporator (W)
W_pump=out.simoutpump; %Pump work (W)

%Interior flow
Dh=repelem(out.simoutDh,fp)';
Re_int=repelem(out.simoutRint,fp)';
f_int=repelem(out.simoutfint,fp)';
k_int=repelem(out.simoutkint,fp)';
mi_int=repelem(out.simoutmiint,fp)';
cp_int=repelem(out.simoutcpint,fp)';
rho_int=repelem(out.simoutrhoint,fp)';
V_int=repelem(out.simoutvint,fp)';
flowrate_int=repelem(out.simoutflowrateint,fp)';
```

```
%Exterior flow
Taw= out.simoutTaw;
Pr_ext=out.simoutPrext;
mi_ext=out.simoutmiext;
cp_ext=out.simoutcpext;
r=out.simoutr;
V_ext=out.simoutvext;
Tenv=out.simouttenv;
rhoenv=out.simoutrhoenv;

%VCS
Q_cond=repelem(out.simoutcond,fp)'; %Heat load in condenser (W)
W_comp=repelem(out.simoutcomp,fp)'; %Compression work (W)

psat_cond=repelem(out.simoutpsatcond,fp)';
psat_evap=repelem(out.simoutpsatevap,fp)';
m_dotref=repelem(out.simoutmdotref,fp)';
T1_VCS=repelem(out.simoutT1VCS,fp)'; %Outlet evaporator temperature (K)/Inlet compressor
temperature (K)
T2s_VCS=repelem(out.simoutT2sVCS,fp)';%Outlet compressor temperature (K)/Inlet condenser
temperature (K) -isentropic
T2_VCS=repelem(out.simoutT2VCS,fp)'; %Outlet compressor temperature (K)/Inlet condenser
temperature (K) -real
T3_VCS=repelem(out.simoutT3VCS,fp)'; %Outlet condenser temperature (K)/Inlet expansion valve
temperature (K)
T4_VCS=repelem(out.simoutT4VCS,fp)';%Outlet expansion valve temperature (K)/Inlet evaporator
temperature (K)
s1_VCS=repelem(out.simouts1VCS,fp)';
s2s_VCS=repelem(out.simouts2sVCS,fp)';
s2_VCS=repelem(out.simouts2VCS,fp)';
s3_VCS=repelem(out.simouts3VCS,fp)';
s4_VCS=repelem(out.simouts4VCS,fp)';
h1_VCS=repelem(out.simouth1VCS,fp)';
h2s_VCS=repelem(out.simouth2sVCS,fp)';
h2_VCS=repelem(out.simouth2VCS,fp)';
h3_VCS=repelem(out.simouth3VCS,fp)';
h4_VCS=repelem(out.simouth4VCS,fp)';
v1_VCS=repelem(out.simoutv1VCS,fp);
v2_VCS=repelem(out.simoutv2VCS,fp);
v3_VCS=repelem(out.simoutv3VCS,fp);
v4_VCS=repelem(out.simoutv4VCS,fp);
%s_VCS=[s1_VCS,s2s_VCS,s2_VCS,s3_VCS,s4_VCS];
%T_VCS=[T1_VCS,T2s_VCS,T2_VCS,T3_VCS,T4_VCS];

Delta_TLMcond=out.simoutDELTATLMcond;
Delta_T1cond=repelem(out.simoutDELTAT1cond,fp)';
Delta_T2cond=out.simoutDELTAT2cond;
UA_cond=out.simoutUAcond;
Delta_TLMevap=out.simoutDTMevap;
Delta_T1evap=out.simoutDT1evap;
Delta_T2evap=repelem(out.simoutDT2evap,fp)';
UA_evap=out.simoutUAevap;

%Ram air inlet
W_fan=[out.simoutfan(end,1),repelem(0,59),out.simoutfan(end,1)]'; %Hydraulic power of fan (W)
m_dotramair=out.simoutmdotramair; %Ram air requires (kg/s)
T_outramair=repelem(out.simouttoutramair,fp)';
T_inramair=out.simouttintramair;


%Effiencies and Performance coeffients
W_comp_elect=W_comp./eta_compressor;
COP=Q_evap./W_comp_elect; %VCS coefficient of performance - COP (W)
W_fan_elect=W_fan./eta_fan; %Electric power consumption of fan (W)
Q_dot_fan=(1-eta_fan).*W_fan_elect; %Heat load fan (W)
Wpump_elect=W_pump./eta_pump;
Q_dot_pump=(1-eta_pump).*Wpump_elect;
```

```
%Energy balance of liquid cooling verification (tolerance due to due to floating points)
B1_LIQUID=Q_equip*10^3;
B2_LIQUID=Q_evap;
if abs(B1_LIQUID-B2_LIQUID)<5*10^(-10)
        disp ('Energy Balance of Liquid cooling verified')
else
    disp('Energy Balance of Liquid cooling NOT verified')
end


%Energy balance of VCS verification
B1_VCS=W_comp+Q_evap;
B2_VCS=Q_cond;
if abs(B1_VCS-B2_VCS)<5*10^(-10)
        disp ('Energy Balance of VCS verified')
else
    disp('Energy Balance of VCS NOT verified')
end


%% Heat Results

Qequip=repelem(Q_equip*10^3,fp)';
figure
subplot(2,2,1);
plot(Qequip,'-s',...
    'LineWidth',2,...
    'MarkerSize',7,...
    'Color','b',...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor','b');
xlabel ('Flight Duration (min)');
ylabel ('Heat Load (W)');

subplot(2,2,2);
plot(Q_evap,'-s',...
    'LineWidth',2,...
    'MarkerSize',7,...
    'Color','b',...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor','b');
xlabel ('Flight Duration (min)');
ylabel (' Evaporator Heat Flux (W)');

subplot(2,2,3);
plot(Q_cond,'-s',...
    'LineWidth',2,...
    'MarkerSize',7,...
    'Color','b',...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor','b');
xlabel ('Flight Duration (min)');
ylabel ('Condenser Heat Flux (W)');

%% Work Results

figure
subplot(2,2,1);
plot(W_comp_elect,'-s',...
    'LineWidth',2,...
    'MarkerSize',7,...
    'Color','b',...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor','b');
xlabel ('Flight Duration (min)');
ylabel ('Compressor electric power consumption (W)');
```

```
subplot(2,2,2);
plot(W_fan_elect,'-s',...
    'LineWidth',2,...
    'MarkerSize',7,...
    'Color','b',...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor','b');
xlabel ('Flight Duration (min)');
ylabel ('Fan electric power consumption (W)');

subplot(2,2,3);
plot(Wpump_elect,'-s',...
    'LineWidth',2,...
    'MarkerSize',7,...
    'Color','b',...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor','b');
xlabel ('Flight Duration (min)');
ylabel ('Pump electric power consumption (W)');

subplot(2,2,4);
plot(m_dotramair,'-s',...
    'LineWidth',2,...
   'MarkerSize',7,...
   'Color','b',...
   'MarkerEdgeColor','b',...
   'MarkerFaceColor','b');
xlabel ('Flight Duration (min)');
ylabel ('Ram air mass flow (kg/s)');

%% Mission profiles
H_60=out.simoutheight;
Mach_60=out.simoutmach;
figure
subplot(1,2,1);
plot(H_60,'-s',...
    'LineWidth',2,...
    'MarkerSize',7,...
    'Color','b',...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor','b');
xlabel ('Flight Duration (min)');
ylabel ('Height (m)');

subplot(1,2,2);
plot(Mach_60,'-s',...
    'LineWidth',2,...
    'MarkerSize',7,...
    'Color','b',...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor','b');
xlabel ('Flight Duration (min)');
ylabel ('Mach number');


%% VCS Temperature vs Entropy
%figure
%plot(s_VCS,T_VCS,'s',...
   % 'LineWidth',2,...
    %'MarkerSize',7,...
    %'Color','b',...
    %'MarkerEdgeColor','b',...
   % 'MarkerFaceColor','b');
%xlabel ('Flight Points');
%ylabel ('W Comp (W)');
```

```
%% Equipment mass
%Data from Embraer

flowrate_ramair=zeros(1,61);
for i=1:61
flowrate_ramair(i)=m_dotramair(i)/rhoenv(i);
end

flowrate_ramair_max=max(flowrate_ramair);


m_pump=max(2*(1942.2*flowrate_int + 1.619)); %different expression
m_condenser=max(0.0008*Q_cond - 0.2165);
m_evaporator=max(0.0005*Q_evap + 2.629);
m_compressor= max(0.001*W_comp + 8.6225);
al_density = 2780; %Alluminum density - SHX material [kg/m³]
cu_density=8960; %Copper density [kg/m^3]


m_liquid_ducts=max(al_density*pi*Dh*L_duct_liquid*t_duct_liquid);
m_liquid=max(rho_int.*pi.*Dh.^2.*L_duct_liquid)/4;

m_ref_ducts=max(cu_density*pi*D_vcs*L_duct_vcs*t_duct_vcs);
rho_R314amax=1/(min([v1_VCS,v2_VCS,v3_VCS,v4_VCS])); %Max density is on stage 3 (liquid)
m_ref=rho_R314amax.*pi.*D_vcs.^2.*L_duct_vcs/4; %use of max density to calculate the
refrigerant masS

m_fan=10.542*flowrate_ramair_max + 0.9366;

m_TMS=2*(m_pump+m_condenser+m_evaporator+m_compressor+m_liquid_ducts+m_liquid+m_ref_ducts+m_r
ef+m_fan);

%% Save to Excel
folder = 'C:\Users\Utente\Desktop\Laurea Magistrale\Thesis\Matlab_model\Technical &
Architectures\Case 1';
baseFileName='Case1TMSResults.xlsx';
fileName = fullfile(baseFileName);


T_in_Equi=repelem(T_InEquip,fp)';


INPUTS=table(Mach_60,H_60,T_in_Equi,Qequip);
writetable(INPUTS,fileName,'Sheet',1)

RESULTS_Liquid=table(Delta_T1evap,Delta_T2evap,Delta_TLMevap,UA_evap,Q_evap,W_pump,Wpump_elec
t,Q_dot_pump);
writetable(RESULTS_Liquid,fileName,'Sheet',2)

RESULTS_VCS=table(psat_evap,m_dotref,T1_VCS,T2_VCS,T2s_VCS,T3_VCS,T4_VCS,s1_VCS,s2_VCS,s2s_VC
S,s3_VCS,s4_VCS,h1_VCS,h2_VCS,h2s_VCS,h3_VCS,h4_VCS,Delta_TLMcond,Delta_T1cond,Delta_T2cond,U
A_cond,Q_cond,W_comp,W_comp_elect,COP,psat_cond);
writetable(RESULTS_VCS,fileName,'Sheet',3)

RESULTS_Ramair=table(m_dotramair,Taw,T_outramair,T_inramair,W_fan,W_fan_elect,Q_dot_fan);
writetable(RESULTS_Ramair,fileName,'Sheet',4)

RESULTS_Mass=table(m_pump,m_evaporator,m_liquid_ducts,m_liquid,m_condenser,m_compressor,m_ref
,m_ref_ducts,m_fan,m_TMS);
writetable(RESULTS_Mass,fileName,'Sheet',5)

%winopen(fileName)
```

Appendix III

- Python4Capella script for the digital connection on logical architecture layer

```
# include needed for the Capella modeller API
include('workspace://Python4Capella/simplified_api/capella.py')
if False:
    from simplified_api.capella import *

# include needed for the PVMT API
include('workspace://Python4Capella/simplified_api/pvmt.py')
if False:
    from simplified_api.pvmt import *

# include needed for utilities
include('workspace://Python4Capella/utilities/CapellaPlatform.py')
if False:
    from utilities.CapellaPlatform import *

# include needed for the requirement API
include('workspace://Python4Capella/simplified_api/requirement.py')
if False:
    from simplified_api.requirement import *

import matlab.engine
eng = matlab.engine.start_matlab()

aird_path = '/TMS_test_6/TMS_test_5.aird'

model = CapellaModel()
model.open(aird_path)

se = model.get_system_engineering()

allLC = se.get_all_contents_by_type(LogicalComponent)

allPVs = []

print('start')

def set_p_v_value(elem, PVName, value):
    for group in elem.get_java_object().getOwnedPropertyValueGroups():
        for pv in group.getOwnedPropertyValues():
            if PVName == pv.getName():
                pv.setValue(value)
                return

for lc in allLC:
    for pvName in PVMT.get_p_v_names(lc):
        if pvName not in allPVs:
            allPVs.append(pvName)


thl=0
m=0
rac=0
pc=0
config='None'

for lc in allLC:
    print(lc.get_name())
    for pvName in allPVs:
        if str(PVMT.get_p_v_value(lc, pvName)) != 'None':
            if pvName=='Configuration':
                config=int(PVMT.get_p_v_value(lc, pvName))
                print('  '+pvName,config)
```

```
            if pvName=='N°':
                n=int(PVMT.get_p_v_value(lc, pvName))
                print('  '+pvName,n)

            if pvName=='Heat load':
                hl=float(PVMT.get_p_v_value(lc, pvName))
                print('  '+pvName,str(hl)+' KW')

            if pvName=='T max':
                T_max=float(PVMT.get_p_v_value(lc, pvName))+float(273) # C° -> K°
                print('  '+pvName,str(T_max)+' K°')

            if pvName=='T min':
                T_min=float(PVMT.get_p_v_value(lc, pvName))+float(273) # C° -> K°
                print('  '+pvName,str(T_min)+' K°')

    if str(config)!= 'None':
        eng.workspace['T_InEquip']=T_min
        eng.workspace['Q_equip']=hl

        if config==1:
            #run simulation

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case1.m',nargout=0)

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case1Results.m',nargout=0)
            mass=eng.workspace['m_TMS']
            ram_air_cons=eng.workspace['mdra']
            power_cons=eng.workspace['w_tot']
            print('  Mass = '+str(round(mass,2))+' kg, Ram air consumption =
'+str(round(ram_air_cons,2))+' kg/s, Power consumption = '+str(round(power_cons,2))+'
W, Configuration: '+str(config))


        elif config==2:

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case2.m',nargout=0)

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case2Results.m',nargout=0)
            mass=eng.workspace['m_TMS']
            ram_air_cons=eng.workspace['mdra']
            power_cons=eng.workspace['w_tot']
            print('  Mass = '+str(round(mass,2))+' kg, Ram air consumption =
'+str(round(ram_air_cons,2))+' kg/s, Power consumption = '+str(round(power_cons,2))+'
W, Configuration: '+str(config))


        elif config==3:

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case4.m',nargout=0)

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case4Results.m',nargout=0)
            mass=eng.workspace['m_TMS']
            ram_air_cons=eng.workspace['mdra']
            power_cons=eng.workspace['w_tot']
            print('  Mass = '+str(round(mass,2))+' kg, Ram air consumption =
'+str(round(ram_air_cons,2))+' kg/s, Power consumption = '+str(round(power_cons,2))+'
W, Configuration: '+str(config))



        elif config==:

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case6.m',nargout=0)

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case6Results.m',nargout=0)
            mass=eng.workspace['m_TMS']
            ram_air_cons=eng.workspace['mdra']
            power_cons=eng.workspace['w_tot']
```

```
            print('  Mass = '+str(round(mass,2))+' kg, Ram air consumption =
'+str(round(ram_air_cons,2))+' kg/s, Power consumption = '+str(round(power_cons,2))+'
W, Configuration: '+str(config))
        thl=thl+hl
        m=m+mass
        rac=rac+ram_air_cons
        pc=pc+power_cons

    config='None'
eng.quit()


# checking requirements
for req in se.get_all_contents_by_type(Requirement):
    #print(str(req.get_id()))
    #print(req.get_long_name())
    incoming_req_names = []
    for res in req.get_incoming_linked_elems():
        incoming_req_names.append(res.get_name())
    outgoing_req_names = []
    for res in req.get_outgoing_linked_elems():
        outgoing_req_names.append(res.get_name())

    if ('TMS for Electrical drive train' in outgoing_req_names)==1:

        if str(req.get_id())=='1':
            # Extracts Requirements Attributes and Values
            if req.get_java_object().getOwnedAttributes() != None:
                for att in req.get_java_object().getOwnedAttributes():
                    #print("- Attribute: "+att.getDefinition().getReqIFLongName()+",
value: "+str(att.getValue()))
                    if str(att.getDefinition().getReqIFLongName())=='Value':
                        max_mass=float(att.getValue())
                        #print(max_mass)

        elif str(req.get_id())=='2':

            # Extracts Requirements Attributes and Values
            if req.get_java_object().getOwnedAttributes() != None:
                for att in req.get_java_object().getOwnedAttributes():
                    #print("- Attribute: "+att.getDefinition().getReqIFLongName()+",
value: "+str(att.getValue()))
                    if str(att.getDefinition().getReqIFLongName())=='Value':
                        max_power=float(att.getValue())
                        #print(max_power)

        elif str(req.get_id())=='3':

            # Extracts Requirements Attributes and Values
            if req.get_java_object().getOwnedAttributes() != None:
                for att in req.get_java_object().getOwnedAttributes():
                    #print("- Attribute: "+att.getDefinition().getReqIFLongName()+",
value: "+str(att.getValue()))
                    if str(att.getDefinition().getReqIFLongName())=='Value':
                        target_spec_pow_diss=float(att.getValue())
                        #print()

        elif str(req.get_id())=='4':

            # Extracts Requirements Attributes and Values
            if req.get_java_object().getOwnedAttributes() != None:
                for att in req.get_java_object().getOwnedAttributes():
                    #print("- Attribute: "+att.getDefinition().getReqIFLongName()+",
value: "+str(att.getValue()))
                    if str(att.getDefinition().getReqIFLongName())=='Value':
                        target_cop=float(att.getValue())
                        #print()

req_mass_perc= (m/max_mass)*100
```

```
req_power_perc= ((pc/1000)/max_power)*100 #KW
spd=thl/m
req_spec_pow_diss=(spd/target_spec_pow_diss)*100
cop=thl/(pc/1000)
req_cop_perc=(cop/target_cop)*100


model.start_transaction()
try:
    for lc in allLC:
        if lc.get_name() == 'TMS for Electrical drive train':
            for pvName in allPVs:
                if pvName=='Mass':
                    set_p_v_value(lc,str(pvName),round(m,2))
                elif pvName=='Power consumption':
                    set_p_v_value(lc,str(pvName),round(pc,2))
                elif pvName=='Ram air consumption':
                    set_p_v_value(lc,str(pvName),round(rac,2))
                elif pvName=='Mass_req':
                    set_p_v_value(lc,str(pvName),round(req_mass_perc,2))
                elif pvName=='Mass_req_bool':
                    if req_cop_perc <= 100:
                        set_p_v_value(lc,str(pvName),'True')
                    else:
                        set_p_v_value(lc,str(pvName),'False')
                elif pvName=='Power_req':
                    set_p_v_value(lc,str(pvName),round(req_power_perc,2))
                elif pvName=='Power_req_bool':
                    if req_cop_perc <= 100:
                        set_p_v_value(lc,str(pvName),'True')
                    else:
                        set_p_v_value(lc,str(pvName),'False')
                elif pvName=='SPD_req':
                    set_p_v_value(lc,str(pvName),round(req_spec_pow_diss,2))
                elif pvName=='SPD_req_bool':
                    if req_cop_perc >= 100:
                        set_p_v_value(lc,str(pvName),'True')
                    else:
                        set_p_v_value(lc,str(pvName),'False')
                elif pvName=='COP_req':
                    set_p_v_value(lc,str(pvName),round(req_cop_perc,2))
                elif pvName=='COP_req_bool':
                    if req_cop_perc >= 100:
                        set_p_v_value(lc,str(pvName),'True')
                    else:
                        set_p_v_value(lc,str(pvName),'False')
except:
    # if something went wrong we rollback the transaction
    model.rollback_transaction()
    raise
else:
    # if everything is ok we commit the transaction
    model.commit_transaction()

print('Total mass = '+str(round(m,2))+' kg, Total ram air consumption =
'+str(round(rac,2))+' kg/s, Total power consumption = '+str(round(pc/1000,2))+' KW')
print('Total heat load dissipated = '+str(round(thl,3))+' KW')
print('Specific power dissipation = '+str(round(spd,3))+' KW/kg')
print('COP = '+str(round(cop,3)))
print('Mass obtained is '+str(round(req_mass_perc,3))+'% to the Maximum Mass')
print('Power obtained is '+str(round(req_power_perc,3))+'% to the Maximum Power')
print('Specific power dissipation obtained is '+str(round(req_spec_pow_diss,3))+'% to
the target Specific power dissipation')
print('COP obtained is '+str(round(req_cop_perc,3))+'% to the COP target')
print('end')
```

Appendix IV

- Heat load per flight phases table

	Description	Duration	Unit	Gas Turbines		Battery			Converters			Distribution			Inverters			eMotors			eBoosters		
				in	out	in	out	dissipated	in	out	dissipated	in	out	dissipated	in	out	dissipated	in	out	dissipated	in	out	dissipated
Electric Power Consumption	Parked		kW																				
	Taxi out	00:02:00	kW	0	0	1.012	931	81	931	885	46	885	876	9	876	832	44	416	374	42	416	374	42
	Take off	00:00:15	kW	9.318	3.261	1.889	1.738	151	1.738	1.651	87	1.651	1.635	16	1.635	1.553	82	777	699	78	777	699	78
	Start of Climb	00:20:31	kW	9.259	3.241	949	873	76	873	829	44	829	821	8	821	780	41	390	351	39	390	351	39
	End of Climb	00:00:00	kW	6.150	2.153	1.170	1.076	94	1.076	1.022	54	1.022	1.012	10	1.012	961	51	481	433	48	481	433	48
	Cruise	01:09:44	kW	5.701	1.995	674	620	54	620	589	31	589	583	6	583	554	29	277	249	28	277	249	28
	Descent	00:23:28	kW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Landing	00:00:12	kW	3.325	1.164	1.573	1.447	126	1.447	1.375	72	1.375	1.361	14	1.361	1.293	68	646	582	64	646	582	64
	Taxi in	00:02:00	kW	0	0	1.012	931	81	931	885	46	885	876	9	876	832	44	416	374	42	416	374	42

Appendix V

- Batteries resistive losses and total thrust on the mission

segment	time [s]	thrust_total [N]	battery_resistive_losses [W]
climb_1	0	35925,58358	39076,18875
climb_1	14,32743424	35759,0983	39231,90381
climb_1	42,98230273	35429,35888	39532,64344
climb_1	57,30973697	35266,11089	39678,11508
climb_2	57,30973697	34836,42796	39688,04557
climb_2	80,9463142	34560,99275	39919,37959
climb_2	128,2194686	34019,32382	40361,11925
climb_2	151,8560459	33753,11331	40572,81653
climb_3	151,8560459	32628,20612	40582,18344
climb_3	182,4043025	32307,76659	40846,96152
climb_3	243,5008158	31680,74158	41353,89834
climb_3	274,0490725	31374,1979	41597,50737
climb_4	274,0490725	30205,90921	41603,31162
climb_4	296,0763469	30002,0109	41775,15561
climb_4	340,1308956	29600,56898	42110,70321
climb_4	362,1581699	29403,03976	42274,68631
climb_5	362,1581699	28556,37683	42277,11399
climb_5	385,7267594	28358,55422	42450,09055
climb_5	432,8639384	27969,47772	42788,47105
climb_5	456,4325279	27778,23931	42954,07077
climb_6	456,4325279	26977,44867	42955,52163
climb_6	481,795433	26785,52132	43131,49251
climb_6	532,5212433	26408,4959	43475,88034
climb_6	557,8841484	26223,41424	43644,43575
climb_7	557,8841484	25455,5522	43645,26222

climb_7	585,3756912	25269,42701	43825,78715
climb_7	640,3587769	24904,3179	44178,81881
climb_7	667,8503197	24725,35116	44351,43104
climb_8	667,8503197	24003,45849	44351,86864
climb_8	697,8889036	23822,92157	44538,18249
climb_8	757,9660713	23469,37881	44901,89757
climb_8	788,0046552	23296,39039	45079,39494
climb_9	788,0046552	22775,7504	45079,58569
climb_9	804,1603901	22688,5356	45174,14021
climb_9	836,4718598	22516,07675	45360,79648
climb_9	852,6275947	22430,835	45452,91119
climb_9_5	852,6275947	22094,50016	45452,95706
climb_9_5	869,6535355	22008,66739	45549,37435
climb_9_5	903,7054172	21839,04	45739,56178
climb_9_5	920,731358	21755,24766	45833,3464
climb_10	920,731358	21263,64173	45833,34109
climb_10	957,8303684	21093,90923	46035,44007
climb_10	1032,028389	20763,06997	46427,54871
climb_10	1069,127399	20601,97816	46617,69466
climb_11	1069,127399	19986,59165	46617,37604
climb_11	1111,277826	19821,91386	46829,79672
climb_11	1195,578678	19501,9956	47239,76871
climb_11	1237,729104	19346,76496	47437,39241
climb_12	1237,729104	18752,44223	47436,07071
climb_12	1286,724774	18592,69046	47660,61702
climb_12	1384,716113	18283,71061	48088,38135
climb_12	1433,711782	18134,48176	48290,75261
cruise	1433,711782	14248,3831	89,84216828
cruise	2198,766462	14216,38417	90,08908953
cruise	3728,87582	14152,74728	90,56609812
cruise	4493,9305	14121,11127	90,79680335
descent_1	4493,9305	1038,691195	0,293803118
descent_1	4520,458848	1191,535126	0,293804236
descent_1	4573,515544	1491,623417	0,293806471
descent_1	4600,043892	1638,995653	0,293807589
descent_2	4600,043892	1648,00291	0,293807589
descent_2	4629,182091	1806,561825	0,293808816
descent_2	4687,45849	2117,262451	0,293811271
descent_2	4716,596689	2269,548839	0,293812499
descent_3	4716,596689	264,7669992	0,293812499
descent_3	4743,179398	428,4586417	0,293813618
descent_3	4796,344815	749,7745348	0,293815858
descent_3	4822,927524	907,5259038	0,293816978
descent_4	4822,927524	892,6594881	0,293816978
descent_4	4852,03411	1062,271823	0,293818204
descent_4	4910,247281	1394,577046	0,293820656
descent_4	4939,353867	1557,414011	0,293821882

descent_5	4939,353867	1564,331791	0,293821882
descent_5	4971,444174	1739,972012	0,293823233
descent_5	5035,624788	2083,322183	0,293825936
descent_5	5067,715094	2251,197943	0,293827288
descent_6	5067,715094	1440,56516	0,293827288
descent_6	5100,515395	1622,060672	0,293828669
descent_6	5166,115997	1976,643855	0,293831432
descent_6	5198,916298	2149,903311	0,293832813
descent_7	5198,916298	2176,244838	0,293832813
descent_7	5235,452401	2364,185888	0,293834352
descent_7	5308,524609	2730,329599	0,293837429
descent_7	5345,060713	2908,735425	0,293838967
descent_8	5345,060713	2958,586053	0,293838967
descent_8	5386,211282	3153,199032	0,2938407
descent_8	5468,512421	3531,028445	0,293844166
descent_8	5509,66299	3714,491071	0,293845898
descent_9	5509,66299	3789,488406	0,293845898
descent_9	5556,700868	3991,168222	0,293847879
descent_9	5650,776623	4380,97923	0,29385184
descent_9	5697,814501	4569,418728	0,29385382
descent_10	5697,814501	4701,933229	0,29385382
descent_10	5783,920061	5025,986383	0,293857445
descent_10	5956,13118	5634,349541	0,293864694
descent_10	6042,23674	5920,112642	0,293868319
descent_11	6042,23674	5800,363171	0,293868319
descent_11	6086,60215	6055,340555	0,293870186
descent_11	6175,332971	6556,620814	0,293873921
descent_11	6219,698382	6803,203023	0,293875788
descent_12	6219,698382	5441,526595	0,293875788
descent_12	6246,231458	5507,260007	0,293876905
descent_12	6299,29761	5644,122369	0,293879139
descent_12	6325,830686	5715,237047	0,293880255
reserve_climb_1	6325,830686	32483,43024	90,82878309
reserve_climb_1	6341,317411	32311,75746	90,83339013
reserve_climb_1	6372,29086	31972,32229	90,84259867
reserve_climb_1	6387,777585	31804,56129	90,84720019
reserve_climb_2	6387,777585	30526,21177	90,84719777
reserve_climb_2	6415,510398	30249,93971	90,85543335
reserve_climb_2	6470,976024	29708,27738	90,87188711
reserve_climb_2	6498,708837	29442,89712	90,88010535
reserve_climb_3	6498,708837	28467,4772	90,88010081
reserve_climb_3	6535,117774	28145,66393	90,89088139
reserve_climb_3	6607,935646	27518,65338	90,9124134
reserve_climb_3	6644,344582	27213,46898	90,92316501
reserve_climb_4	6644,344582	26206,98641	90,92315868
reserve_climb_4	6671,0846	26002,39864	90,93104894
reserve_climb_4	6724,564636	25600,96179	90,9468143

reserve_climb_4	6751,304654	25404,11696	90,95468947
reserve_climb_5	6751,304654	24678,11119	90,95468461
reserve_climb_5	6780,41441	24479,51482	90,96325201
reserve_climb_5	6838,633924	24090,44447	90,98036943
reserve_climb_5	6867,743681	23899,97367	90,98891957
reserve_cruise	6867,743681	12013,50277	90,98891401
reserve_cruise	6990,691175	12003,87341	91,02496444
reserve_cruise	7236,586163	11984,65468	91,09678277
reserve_cruise	7359,533657	11975,06506	91,1325616
reserve_descent_3	7359,533657	1766,907746	0,294618769
reserve_descent_3	7395,781626	1960,843083	0,294620292
reserve_descent_3	7468,277563	2338,694389	0,29462334
reserve_descent_3	7504,525532	2522,812764	0,294624863
reserve_descent_4	7504,525532	2560,735663	0,294624863
reserve_descent_4	7545,309664	2761,517551	0,294626578
reserve_descent_4	7626,87793	3151,359741	0,294630007
reserve_descent_4	7667,662063	3340,665984	0,294631721
reserve_descent_5	7667,662063	3423,462166	0,294631721
reserve_descent_5	7740,294559	3744,425072	0,294634774
reserve_descent_5	7885,55955	4352,964241	0,294640882
reserve_descent_5	7958,192046	4641,641579	0,294643935
reserve_descent_6	7958,192046	4762,396704	0,294643935
reserve_descent_6	8032,856808	5040,062634	0,294647075
reserve_descent_6	8182,186333	5565,784534	0,294653354
reserve_descent_6	8256,851096	5814,759511	0,294656493
reserve_descent_7	8256,851096	5457,858048	0,294656493
reserve_descent_7	8283,384172	5525,173773	0,294657609
reserve_descent_7	8336,450324	5665,137885	0,294659841
reserve_descent_7	8362,9834	5737,772697	0,294660956
hold	8362,9834	11891,61422	91,14966114
hold	8812,9834	11855,63484	91,27988386
hold	9712,9834	11784,27528	91,53922358
hold	10162,9834	11748,89312	91,66987021

Appendix VI

- Python4Capella script for the digital connection on physical architecture layer

```
# include needed for the Capella modeller API
include('workspace://Python4Capella/simplified_api/capella.py')
if False:
    from simplified_api.capella import *

# include needed for the PVMT API
include('workspace://Python4Capella/simplified_api/pvmt.py')
if False:
    from simplified_api.pvmt import *

# include needed for utilities
include('workspace://Python4Capella/utilities/CapellaPlatform.py')
if False:
    from utilities.CapellaPlatform import *

# include needed for the requirement API
include('workspace://Python4Capella/simplified_api/requirement.py')
if False:
    from simplified_api.requirement import *

import matlab.engine
eng = matlab.engine.start_matlab()

aird_path = '/Transition_test_2/Transition_test_2.aird'

model = CapellaModel()
model.open(aird_path)

se = model.get_system_engineering()

allPC = se.get_all_contents_by_type(NodePC)

allPVs = []

print('start\n')

def set_p_v_value(elem, PVName, value):
    for group in elem.get_java_object().getOwnedPropertyValueGroups():
        for pv in group.getOwnedPropertyValues():
            if PVName == pv.getName():
                pv.setValue(value)
                return

for pc in allPC:
    for pvName in PVMT.get_p_v_names(pc):
        if pvName not in allPVs:
            allPVs.append(pvName)

ns='REQUIREMENT NOT SATISFIED'
s='REQUIREMENT SATISFIED'
thl=0
m=0
rac=0
pcs=0
config=0
hl=0
flag=0
n=0
output=0
req1=0
req2=0
req3=0
req4=0

while output < 1 or output > 3:
    def showInputDialog():
        pass
```

```
    loadModule('/System/UI')

    output = int(showInputDialog('Select console output type\n1 - All results\n2 - Verification
results\n3 - No results\n\n(In any case all results will be placed in the model)', 'Results type
number'))


while n < 1 or n > 9:
    def showInputDialog():
        pass

    loadModule('/System/UI')


    n = int(showInputDialog('Select flight phase number\nFlight phases:\n1 - Parked\n2 - Taxi
out\n3 - Take-off\n4 - Start of climb\n5 - End of climb\n6 - Cruise\n7 - Descent\n8 - Landing\n9
- Taxi in', 'Flight phase number'))


if n==1:
    phase='Parked'
elif n==2:
    phase='Taxi out'
elif n==3:
    phase='Take-off'
elif n==4:
    phase='Start of climb'
elif n==5:
    phase='End of climb'
elif n==6:
    phase='Cruise'
elif n==7:
    phase='Descent'
elif n==8:
    phase='Landing'
elif n==9:
    phase='Taxi in'

print('TMS sizing on "'+str(phase)+'" conditions\n')


for pc in allPC:

    for pvName in allPVs:
        if str(PVMT.get_p_v_value(pc, pvName)) != 'None':
            if pvName=='Configuration':
                config=int(PVMT.get_p_v_value(pc, pvName))


            if pvName=='T':
                T=float(PVMT.get_p_v_value(pc, pvName))+float(273) # C° -> K°



    if isinstance(config,int) and int(config) > 0:
        print(pc.get_name())
        eng.workspace['T_InEquip']=T

        sms=pc.get_owned_state_machines()
        for sm in sms:
            regions=sm.get_owned_regions()
            for region in regions:
                states = region.get_owned_states()
                for state in states:
                    rs=state.get_owned_regions()
                    for r in rs:
                        if state.get_name()==phase:
                            hl=float(r.get_name())

        print('  Configuration '+str(config))
        print('  T = '+str(T)+' °K')
        print('  Heat load = '+str(hl)+' KW')
        eng.workspace['Q_equip']= float(hl)

        if hl > 0:
            flag=1
```

```
            if config==1:
                #run simulation
                eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case1.m',nargout=0)

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case1Results.m',nargout=0)
                mass=eng.workspace['m_TMS']
                ram_air_cons=eng.workspace['mdra']
                power_cons=eng.workspace['w_tot']
                if power_cons > 0:
                    print('  Mass = '+str(round(mass,2))+' kg, Ram air consumption =
'+str(round(ram_air_cons,2))+' kg/s, Power consumption = '+str(round(power_cons,2))+' W\n')
                else:
                    mass=0
                    ram_air_cons=0
                    power_cons=0
                    print('  The evaluated power consumption is < 0, cooling not required for
'+str(pc.get_name())+' in '+str(phase)+' phase')

            elif config==2:
                eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case2.m',nargout=0)

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case2Results.m',nargout=0)
                mass=eng.workspace['m_TMS']
                ram_air_cons=eng.workspace['mdra']
                power_cons=eng.workspace['w_tot']
                if power_cons > 0:
                    print('  Mass = '+str(round(mass,2))+' kg, Ram air consumption =
'+str(round(ram_air_cons,2))+' kg/s, Power consumption = '+str(round(power_cons,2))+' W\n')
                else:
                    mass=0
                    ram_air_cons=0
                    power_cons=0
                    print('  The evaluated power consumption is < 0, cooling not required for
'+str(pc.get_name())+' in '+str(phase)+' phase')


            elif config==3:
                eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case4.m',nargout=0)

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case4Results.m',nargout=0)
                mass=eng.workspace['m_TMS']
                ram_air_cons=eng.workspace['mdra']
                power_cons=eng.workspace['w_tot']
                if power_cons > 0:
                    print('  Mass = '+str(round(mass,2))+' kg, Ram air consumption =
'+str(round(ram_air_cons,2))+' kg/s, Power consumption = '+str(round(power_cons,2))+' W\n')
                else:
                    mass=0
                    ram_air_cons=0
                    power_cons=0
                    print('  The evaluated power consumption is < 0, cooling not required for
'+str(pc.get_name())+' in '+str(phase)+' phase')



            elif config==4:
                eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case6.m',nargout=0)

eng.run('C:/Users/Utente/PycharmProjects/pythonProject7/Case6Results.m',nargout=0)
                mass=eng.workspace['m_TMS']
                ram_air_cons=eng.workspace['mdra']
                power_cons=eng.workspace['w_tot']
                if power_cons > 0:
                    print('  Mass = '+str(round(mass,2))+' kg, Ram air consumption =
'+str(round(ram_air_cons,2))+' kg/s, Power consumption = '+str(round(power_cons,2))+' W\n')
                else:
                    mass=0
                    ram_air_cons=0
                    power_cons=0
                    print('  The evaluated power consumption is < 0, cooling not required for
'+str(pc.get_name())+' in '+str(phase)+' phase')

            thl=thl+hl
            m=m+mass
            rac=rac+ram_air_cons
            pcs=pcs+power_cons
```

```
        else:
            print('  Cooling not required for '+str(pc.get_name())+' in '+str(phase)+' phase,
heat load = '+str(hl))

    config=0

if pcs == 0:
    flag=0

eng.quit()

if flag==1:
    for req in se.get_all_contents_by_type(Requirement):
        #print(str(req.get_id()))
        #print(req.get_long_name())
        incoming_req_names = []
        for res in req.get_incoming_linked_elems():
            incoming_req_names.append(res.get_name())
        outgoing_req_names = []
        for res in req.get_outgoing_linked_elems():
            outgoing_req_names.append(res.get_name())

        if ('TMS for electrical drive train' in outgoing_req_names)==1:

            if str(req.get_id())=='1':
                # Extracts Requirements Attributes and Values
                if req.get_java_object().getOwnedAttributes() != None:
                    for att in req.get_java_object().getOwnedAttributes():
                        #print("- Attribute: "+att.getDefinition().getReqIFLongName()+", value:
"+str(att.getValue()))
                        if str(att.getDefinition().getReqIFLongName())=='Value':
                            max_mass=float(att.getValue())
                            #print(max_mass)

            elif str(req.get_id())=='2':

                # Extracts Requirements Attributes and Values
                if req.get_java_object().getOwnedAttributes() != None:
                    for att in req.get_java_object().getOwnedAttributes():
                        #print("- Attribute: "+att.getDefinition().getReqIFLongName()+", value:
"+str(att.getValue()))
                        if str(att.getDefinition().getReqIFLongName())=='Value':
                            max_power=float(att.getValue())
                            #print(max_power)

            elif str(req.get_id())=='3':

                # Extracts Requirements Attributes and Values
                if req.get_java_object().getOwnedAttributes() != None:
                    for att in req.get_java_object().getOwnedAttributes():
                        #print("- Attribute: "+att.getDefinition().getReqIFLongName()+", value:
"+str(att.getValue()))
                        if str(att.getDefinition().getReqIFLongName())=='Value':
                            target_spec_pow_diss=float(att.getValue())
                            #print(target_spec_pow_diss)

            elif str(req.get_id())=='4':

                # Extracts Requirements Attributes and Values
                if req.get_java_object().getOwnedAttributes() != None:
                    for att in req.get_java_object().getOwnedAttributes():
                        #print("- Attribute: "+att.getDefinition().getReqIFLongName()+", value:
"+str(att.getValue()))
                        if str(att.getDefinition().getReqIFLongName())=='Value':
                            target_cop=float(att.getValue())
                            #print(target_cop)

    req_mass_perc= (m/max_mass)*100
    req_power_perc= ((pcs/1000)/max_power)*100 #KW
    spd=thl/m
    req_spec_pow_diss=(spd/target_spec_pow_diss)*100
    cop=thl/(pcs/1000)
    req_cop_perc=(cop/target_cop)*100


    model.start_transaction()
    try:
```

```
for pc in allPC:
    if pc.get_name() == 'TMS for electrical drive train':
        for pvName in allPVs:
            if pvName=='Mass':
                set_p_v_value(pc,str(pvName),round(m,2))
            elif pvName=='Power_consumption':
                set_p_v_value(pc,str(pvName),round(pcs,2))
            elif pvName=='Ram_air_consumption':
                set_p_v_value(pc,str(pvName),round(rac,2))
            elif pvName=='Mass_req':
                set_p_v_value(pc,str(pvName),round(req_mass_perc,2))
            elif pvName=='Mass_req_bool':
                if req_mass_perc<= 100:
                    set_p_v_value(pc,str(pvName),'True')
                    req1=1
                else:
                    set_p_v_value(pc,str(pvName),'False')
            elif pvName=='Power_req':
                set_p_v_value(pc,str(pvName),round(req_power_perc,2))
            elif pvName=='Power_req_bool':
                if req_power_perc <= 100:
                    set_p_v_value(pc,str(pvName),'True')
                    req2=1
                else:
                    set_p_v_value(pc,str(pvName),'False')
            elif pvName=='SPD_req':
                set_p_v_value(pc,str(pvName),round(req_spec_pow_diss,2))
            elif pvName=='SPD_req_bool':
                if req_spec_pow_diss >= 100:
                    set_p_v_value(pc,str(pvName),'True')
                    req3=1
                else:
                    set_p_v_value(pc,str(pvName),'False')
            elif pvName=='COP_req':
                set_p_v_value(pc,str(pvName),round(req_cop_perc,2))
            elif pvName=='COP_req_bool':
                if req_cop_perc >= 100:
                    set_p_v_value(pc,str(pvName),'True')
                    req4=1
                else:
                    set_p_v_value(pc,str(pvName),'False')
            elif pvName=='Run_result':
                set_p_v_value(pc, pvName,'TMS evaluated')
            elif pvName=='Run_phase':
                set_p_v_value(pc,pvName,phase)

for req in se.get_all_contents_by_type(Requirement):
#print(str(req.get_id()))
#print(req.get_long_name())
    incoming_req_names = []
    for res in req.get_incoming_linked_elems():
        incoming_req_names.append(res.get_name())
    outgoing_req_names = []
    for res in req.get_outgoing_linked_elems():
        outgoing_req_names.append(res.get_name())

    if ('TMS for electrical drive train' in outgoing_req_names)==1:

        if str(req.get_id())=='1':
            if req1==0:
                req.set_prefix(ns)

            else:
                req.set_prefix(s)

        if str(req.get_id())=='2':
            if req2==0:
                req.set_prefix(ns)

            else:
                req.set_prefix(s)


        if str(req.get_id())=='3':
            if req3==0:
                req.set_prefix(ns)

            else:
```

```
                        req.set_prefix(s)


                if str(req.get_id())=='4':
                    if req4==0:
                        req.set_prefix(ns)

                    else:
                        req.set_prefix(s)


    except:
        # if something went wrong we rollback the transaction
        model.rollback_transaction()
        raise
    else:
        # if everything is ok we commit the transaction
        model.commit_transaction()

    print('\n---------RESULTS--------\n')

    if output==1:
        print('Total mass = '+str(round(m,2))+' kg, Total ram air consumption =
'+str(round(rac,2))+' kg/s, Total power consumption = '+str(round(pcs/1000,2))+' KW')
        print('Total heat load dissipated = '+str(round(thl,3))+' KW')
        print('Specific power dissipation = '+str(round(spd,3))+' KW/kg')
        print('COP = '+str(round(cop,3)))
        print('\n')
        if req1==0:
            print('Mass obtained is '+str(round(req_mass_perc,3))+'% to the Maximum Mass, '+ns)
        else:
            print('Mass obtained is '+str(round(req_mass_perc,3))+'% to the Maximum Mass, '+s)
        if req2==0:
            print('Power obtained is '+str(round(req_power_perc,3))+'% to the Maximum Power,
'+ns)
        else:
            print('Power obtained is '+str(round(req_power_perc,3))+'% to the Maximum Power,
'+s)
        if req3==0:
            print('Specific power dissipation obtained is '+str(round(req_spec_pow_diss,3))+'%
to the target Specific power dissipation, '+ns)
        else:
            print('Specific power dissipation obtained is '+str(round(req_spec_pow_diss,3))+'%
to the target Specific power dissipation, '+s)

        if req4==0:
            print('COP obtained is '+str(round(req_cop_perc,3))+'% to the COP target, '+ns)
        else:
            print('COP obtained is '+str(round(req_cop_perc,3))+'% to the COP target, '+s+'\n')

    elif output==2:
        if req1==0:
            print('Mass '+ns)
        else:
            print('Mass '+s)
        if req2==0:
            print('Power '+ns)
        else:
            print('Power '+s)
        if req3==0:
            print('Specific power dissipation '+ns)
        else:
            print('Specific power dissipation '+s)

        if req4==0:
            print('COP '+ns)
        else:
            print('COP '+s+'\n')
    elif output==3:
        print('All results in the model\n')
else:
    print(str(phase)+' does not require cooling, TMS not evaluated')
    model.start_transaction()
    try:
        for pc in allPC:
            if pc.get_name() == 'TMS for electrical drive train':
                for pvName in allPVs:
                    if pvName=='Mass':
```

```
                    set_p_v_value(pc,str(pvName),float(0))
                elif pvName=='Power_consumption':
                    set_p_v_value(pc,str(pvName),float(0))
                elif pvName=='Ram_air_consumption':
                    set_p_v_value(pc,str(pvName),float(0))
                elif pvName=='Mass_req':
                    set_p_v_value(pc,str(pvName),float(0))
                elif pvName=='Mass_req_bool':
                    set_p_v_value(pc,str(pvName),'-')

                elif pvName=='Power_req':
                    set_p_v_value(pc,str(pvName),float(0))
                elif pvName=='Power_req_bool':
                    set_p_v_value(pc,str(pvName),'-')
                elif pvName=='SPD_req':
                    set_p_v_value(pc,str(pvName),float(0))
                elif pvName=='SPD_req_bool':
                    set_p_v_value(pc,str(pvName),'-')
                elif pvName=='COP_req':
                    set_p_v_value(pc,str(pvName),float(0))
                elif pvName=='COP_req_bool':
                    set_p_v_value(pc,str(pvName),'-')
                elif pvName=='Run_result':
                    set_p_v_value(pc, pvName,'TMS not evaluated')
                elif pvName=='Run_phase':
                    set_p_v_value(pc, pvName,phase)
    except:
        # if something went wrong we rollback the transaction
        model.rollback_transaction()
        raise
    else:
        # if everything is ok we commit the transaction
        model.commit_transaction()

print('end')
```

Bibliography

(1) Pascal Roques, "Systems Architecture Modeling with the Arcadia Method", Elsevier, 2018.

(2) https://www.reusecompany.com/rat-authoring-tools

(3) https://www.reusecompany.com/v-v-studio-verification-and-validation

(4) https://www.reusecompany.com/km-knowledge-manager

(5) https://mbseworks.com/mbse-faq/how-does-mbse-enable-requirements-verification-and-validation.html

(6) https://www.incose.org/incose-member-resources/working-groups/process/requirements

(7) https://www.maplesoft.com/products/maplembse.https://www.ibm.com/docs/it/ermd/9.7.0?topic=overview-doors

(8) https://forum.mbse-capella.org/t/how-to-transform-capella-model-into-matlabs-simulink-model/5333

(9) http://www.pgmse.com/dess

(10) Paul R. Montgomery, "Model-Based System Integration (MBSI) – Key Attributes of MBSE from the System Integrator's Perspective", Elsevier, 2013.

(11) Pascal Graignic, Thomas Vosgien, Marija Jankovic, Vincent Tuloup, Jennifer Berquet, Nadège Troussier, "Complex System Simulation: Proposition of a MBSE Framework for Design-Analysis Integration", Elsevier 2013.

(12) https://it.mathworks.com/

(13) M. Chami, A. Aleksandraviciene, A. Morkevicius, and J.-M. Bruel, "Towards Solving MBSE Adoption Challenges: The D3 MBSE Adoption Toolbox", INCOSE.

(14) A. Butting, S. Konar, B. Rumpe, and A. Wortmann, "Teaching model-based systems engineering for industry 4.0: Student challenges and expectations", in 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS-Companion 2018, Oct. 2018, pp. 74–81, doi: 10.1145/3270112.3270122.

(15) D. D. Walden, G. J. Roedler, K. J. Forsberg, R. D. Hamelin, and T. M. Shortell, "Systems Engineering Handbook", 2015.

(16) E. E. Bürger, "A CONCEPTUAL MBSE FRAMEWORK FOR SATELLITE AIT PLANNING", 2019.

(17) M. Ingham et al., "A Model-Based Approach to Engineering Behavior of Complex Aerospace Systems".

(18) L. E. Hart Lockheed Martin and G. LauraEHart, "Introduction to Model-Based System Engineering (MBSE) and SysML", 2015.

(19) S. P. Alai, "EVALUATING ARCADIA/CAPELLA VS. OOSEM/SYSML FOR SYSTEM ARCHITECTURE DEVELOPMENT", 2019.

(20) J. A. Estefan, "Survey of Model-Based Systems Engineering (MBSE) Methodologies", 2008.

(21) A. Alberts and C. Zingel, "Challenges of Model-Based Systems Engineering: A Study towards Unified Term Understanding and the State of Usage of SysML", 2013.

(22) A. Morkevicius, A. Aleksandraviciene, D. Mazeika, L. Bisikirskiene, and Z. Strolia, “MBSE Grid: A Simplified SysML-Based Approach for Modeling Complex Systems”, 2017.
(23) D. Eisenhut, N. Moebs, E. Windels, D. Bergmann, I. Geiß, R. Reis, A. Strohmayer, “Aircraft Requirements for Sustainable Regional Aviation”, Aerospace, 2021.
(24) https:// www . statista . com / statistics / 564769 / airline - industry - number - of - flights/.
(25) https://aviationbenefits.org/media/167143/abbb20_full.pdf.
(26) https://www.icao.int/environmental-protection/Documents/ICAO-ENV-Report2019-F1-WEB%20(1).pdf.
(27) https://futprint50.eu/
(28) S. Jackson, “System Engineering for Commercial Aircrafts”, Ashgate, 1997.
(29) Ian Moir, Allan G. Seabridge, “Civil Avionics Systems”, John Wiley & Sons, Ltd.
(30) D. Vertemati, “Preliminary study on tools to enable the connection between Capella and Matlab”, Politecnico di Torino, 2022.
(31) https://forum.mbse-capella.org/
(32) Help - Capella
(33) https://it.mathworks.com/
(34) https://www.python.org/
(35) E. Brusa, A. Calà, D. Ferretto, “Systems Engineering and Its Application to Industrial Product Development, Springer, 2018.