Global Model of Aircraft Design: from Performance Requirements towards Architectures Optimization

Carlos Garcia-Rubio, Kittinan Thanissaranon, Jean-Charles Chaudemar, Nathalie Bartoli, Thierry Lefebvre

AeroBest 2021
Lisboa, 21-23 July 2021

GLOBAL MODEL OF AIRCRAFT DESIGN: FROM PERFORMANCE REQUIREMENTS TOWARDS ARCHITECTURES OPTIMIZATION

Carlos GARCIA-RUBIO[1*], Kittinan THANISSARANON[1*], Jean-Charles CHAUDEMAR[2], Nathalie BARTOLI[3] and Thierry LEFEBVRE[3]

1: ISAE-SUPAERO, Université de Toulouse
Toulouse, France
{Carlos.Garcia-Rubio,Kittinan.Thanissaranon}@student.isae-supaero.fr

2: ISAE-SUPAERO, Université de Toulouse
Toulouse, France
jean-charles.chaudemar@isae-supaero.fr

3: ONERA/DTIS, Université de Toulouse
Toulouse, France
{Nathalie.Bartoli,Thierry.Lefebvre}@onera.fr

Abstract. *Systems engineering is a transdisciplinary approach that seeks the successful realisation of a system. In order to reach that, it is necessary to satisfy a series of needs and stakeholders, which have to be defined as a group of requirements to be fulfilled. Design approaches such as MDAO, MBSE, and MBSA have been created to help better designing aircraft in different aspects. However, there has never been an aircraft design method which combines all the mentioned design approaches and that, nowadays, has an increasing interest in the industry. Therefore, the paper aim is to produce aircraft design tools and methodology which link all the three design approaches together by using a surveillance UAV as a study case. As a part of the MDAO-MBSE global project, a MDAO model has been developed using a web application (WhatsOpt) and a MDAO platform (OpenMDAO) on Python, while the tool considered to develop the MBSE model is Eclipse Papyrus, a powerful tool that allows to use XML for export their models. Some promising results show the coupling between MDAO and MBSE based on XML file exchange.*

Keywords: MBSE, MDAO, SysML, XML

1 INTRODUCTION

Aircraft design is a complex process that involves several fields of study. Currently, there are different aircraft design approaches which help engineers designing aircraft more effectively. Approaches which are widely and separately used are MDAO (Multidisciplinary Design Analysis and Optimization process), MBSE (Model-Based Systems Engineering), and MBSA (Model-Based Safety Assessment). MDAO focuses on optimizing and obtaining design parameters to achieve certain design objectives. MBSE focuses on the traceability of design requirements and the consistency of information throughout all engineering processes. Finally, MBSA focuses on the safety analysis of a system.

Up to today, there has not been work which combines all the three design approaches together to effectively design an aircraft. Given that, the goal of this study is to identify and obtain a framework which includes MDAO, MBSE, and MBSA to optimally design an aircraft while meeting all the safety objectives by using a surveillance UAV as a studying subject. Given that, this project was initially divided into two parts: MDAO-MBSE part and MBSA-MBSE part. This reports focuses on the MDAO-MBSE part. Firstly, a brief explanation of MDAO and MBSE is provided along with the study-subject UAV and the developed MDAO model from the work earlier. Then, the investigation method used for developing MBSE models is given. The investigation method includes requirements implementation on MBSE, MDAO implementation on Eclipse Papyrus, and MDAO-MBSE connection. After that, the result and analysis obtained from the investigation method are explained. Finally, the conclusion of the project along with the perspectives for future work are provided.

2 MDAO AND MBSE MODELS AND CONTEXT

2.1 MDAO and XDSM

Since designing an aircraft involves several disciplines such as aerodynamics, structure, propulsion, and performance, any design changes in one discipline certainly will affect other disciplines. MDAO allows the possibility to obtain the optimized value of the desired parameters in order to minimize some quantities of interest such as fuel consumption or weight with respect to some constraints. MDAO models are usually displayed in a graphical representation known as eXtended Design Structure Matrix (XDSM). Figure 1 shows an example of an XDSM of a system that has four disciplines.

While green rectangle represents each discipline, grey parallelogram represents response variables which are being transferred and calculated among the disciplines. White parallelogram represents design variables which are initially introduced to the system. Finally, the round rectangular shape represents driver which distributes variables and performs the optimization calculation.

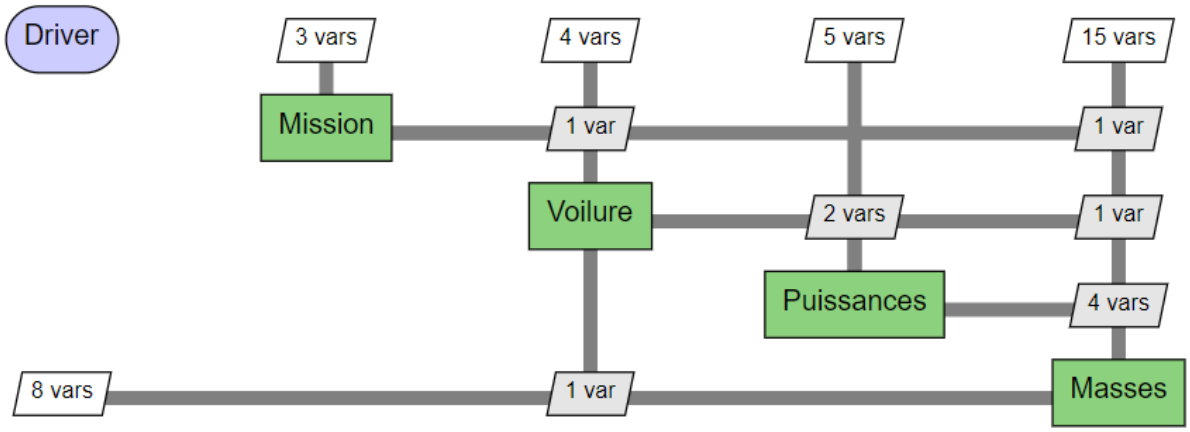


Figure 1: MDAO Mock-up model [1].

Within this project, there are two tools that were used to obtain the MDAO model of a UAV. The first tool is OpenMDAO [2]. It is an opensource platform on Python that is capable of performing an optimization calculation. The second tool is WhatsOpt [3]. It is a web-based tool that allows users to graphically construct an XDSM [4] representation, which can be exported directly from a web browser to the computer as Python files. The exported files are to be further coded on Python along with the previously mentioned OpenMDAO platform.

2.2 MBSE

The more complex a system is, the more difficult to trace down the design requirements and also to ensure that the information is consistent among each design process. The MBSE approach replaces the previous document-based approach.

Initially, MBSE used to propose methods close to software engineering to visualize the design of a system using UML (Unified Modeling Language). Then, due to the absence of specific MBSE concepts, SysML (System Modeling Language), an extension of the UML, was developed [5]. SysML provides nine different kinds of diagrams displaying different perspectives of a system. Within the scope of this project, Requirements diagram and Block Definition Diagram (BDD) were mainly used.

Apart from allowing the visualization of relationship among each requirement, Requirements diagram also enables the traceability of requirements to other elements in the system (i.e., BDD) [6]. The main elements in the diagram includes Requirement and Requirements relationship. Requirement is basically a block that has two string attributes: “id” and “text” to tag and accommodate a requirement respectively. Requirements relationships show the relationships between each requirement for example: “Derive” and “Refine”. While “Derive” emphasizes that a requirement is further expanded, “Refine” means a requirement is further clarified.

BDD provides the capability to visualize the structural architecture of a system. The important elements that were used within the scope of this project are Block, Port and ItemFlow. A Block contains Property which is practically its attribute. Port can be used on each Block to specify the Property being exchanged among the Blocks. Lastly, ItemFlow specifies the path of the Property exchange between Ports.

Within this project, Eclipse Papyrus with SysML 1.6 was used to house the MBSE

models. It is an opensource tool that supports UML/SysML modeling with the capability to have a semantic variation to customize diagrams and Object Constraint Language (OCL) to implement a consistency validation process.

Study case

To develop a framework that includes the MDAO, MBSE, and MBSA model, a study subject was needed. Inspired by Delair's DT26 [7], missions and certain parameters of the study subject were referred to.

2.3 Context and key issues

Initially, the MDAO model was first developed. Starting with a simple UAV design developed by ONERA [1] as seen in Fig. 1, there are 4 main disciplines: Mission, Voilure, Puissances, and Masses. With the given model, the Voilure discipline was redesigned and renamed to be called Aero. Then the Fuselage and the Empennage disciplines were also added as seen in Fig. 2.

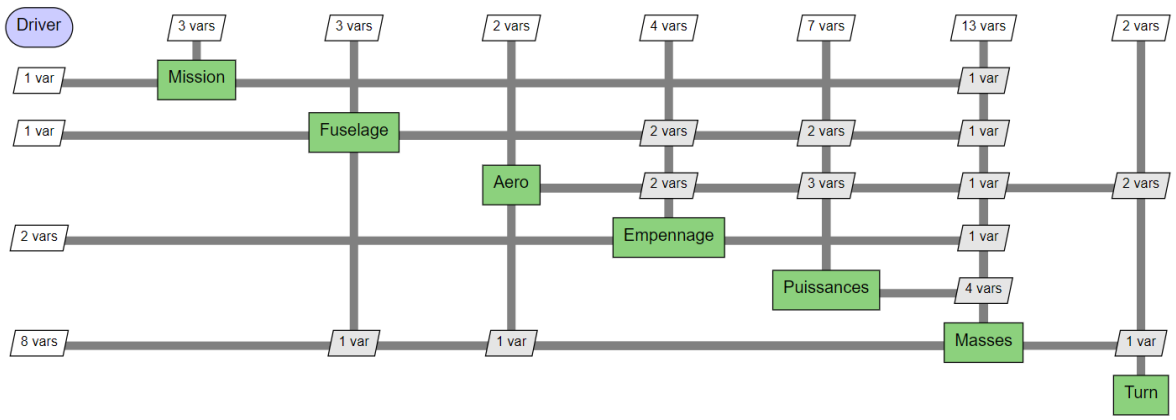


Figure 2: Final MDAO model from last year research.

Each discipline has inputs, outputs, and equations that calculate different design parameters. Section below briefly explains each discipline. Note that the detailed explanation was thoroughly explained in previous reports [8], [9].

Mission

The Mission discipline calculates the vit_max from the defined vit_eco and vit_vent. Then it uses the simple velocity equation to calculate the duree_crois.

Fuselage

The Fuselage discipline calculates the area_fus by assuming that the shape of the fuselage is a cylinder. Then with the assumption of skin friction drag as a majority of drag, Cx_fus is calculated by combining drag from the laminar and turbulent area. Finally, masse_fuselage is calculated from the mass approximation equation [10].

Aero

The Aero discipline calculates the wing parameter such as chord, surface, and Cz using the OpenAeroStruct library and constraint inputs like vit_eco and wing_span.

Empennage

The Empennage discipline calculates S_HT and S_VT from the tail volume equation [11] with the tail volume constant [11]. Also, masse_empenn is calculated from the mass approximation equation [10].

Puissances

The Puissances discipline calculates the puissance_crois_eco and puissance_crois_max from Cx_fus, area_fus, Cx, Cz and surface with the basic drag coefficient and velocity equation.

Masses

The Masses discipline calculates the mass of motor and turbine according to the power needed. It also calculates the empennage mass from the calculated surface area using the historical data. Finally, it combines mass of every component and obtains mtow.

Turn

The Turn discipline calculates the bank angle and the turn radius from the aircraft parameters such as vit_eco, mtow, and surface using the steady turn assumption. The importance of this discipline is that the operating range is constrained by regulations and requirements.

2.4 Aims and objectives

Recall the aim of this project to successfully connect the MDAO and MBSE models together. This means the realization of a design framework which optimizes system parameters (MDAO) while ensuring the consistency and fulfilling all requirements (MBSE).

Since the MDAO part was well established, the steps to be further achieved are to obtain the MBSE part and to identify the connection between MDAO and MBSE. Given that the following objectives were planned.

1. To identify all related UAV's stakeholders and requirements
2. To document all the requirements onto MBSE
3. To establish a connection between MBSE and MDAO
4. To establish information transferring method between MDAO and MBSE

3 INVESTIGATION METHOD

The principal objective of this project is to be able to link the requirements of the surveillance UAV defined in the early stages of any system design (and refined in MBSE) with the design model and its optimization in MDAO as it has been explained. In order to achieve this connection, three steps should be performed: 1) definition of the system requirements and its refining in MBSE, 2) preparation for implementing the MDAO model in Eclipse Papyrus (MBSE) and, 3) the connection between models using XML language.

3.1 Requirements implementation in MBSE

The requirements definition is a process done previously to the architecture design. It gathers all the needs from the stakeholders and transforms them into specific, measurable, achievable and traceable requirements which must be fulfilled by the system after its development.

The first step is, thus, the identification of the different stakeholders involved on the design and use of the UAV and its interest and priority in the system development as it can be seen in Table 1. This will allow to define all the needs and technical requirements of the system which are displayed in [12].

Table 1: Stakeholders list.

ID	Stakeholder	Interest	Flexibility/Priority
STH.01	Shareholders	Profit/Funding	Mandatory / Medium
STH.02	Manufacturer	Design/Performance/Cost/Delay	Mandatory / High
STH.03	Operator	Quality/Performance/Design	Mandatory / High
STH.04	Customer	Quality/Delay/Services	Mandatory / High
STH.05	G.F	Regulation/Employment	Not applicable / Low
STH.05.1	D.G.A	Legislation/Safety	Mandatory / High
STH.06	Local community	Employment/environmental issues	Optional / Medium
STH.07	Environmental association	Pollution/Regulation	Optional / Medium

Profile definition and Requirement diagram in SysML

Once all the requirements are defined, the next step is to introduce them in the MBSE model. In order to do so, Eclipse Papyrus allows to gathered all of them in the Requirement diagram available in the SysML package. This diagram represents the requirements in blocks which contains a descriptive text of them and an identifying id. However, Papyrus and SysML give the possibility of refining this diagram in two different ways.

The first one is creating a stereotyping profile in Papyrus. The creation method of the profile is explained in [12] and it is displayed in Fig. 3. It allows to introduce more information in the requirement block, such as the type (performance, operational context, functional, design or safety requirement), the source (if the requirement was found in any special documentation), the stakeholder's involved and a verified attribute.

The second way is to connect the requirements using the diagram in SysML. Requirement diagram has some features that help to establish links between the requirements, which are generally used to clarify their hierarchy, as it can be seen in Fig. 4. For this project, three different types of link are used: *refine*, used when the client requirement add information more concrete than the one contained in the supplier requirement; *derive*,

which conveys that the requirement at the client end is derived from the requirement at the supplier end (e.g. when the client requirement imposes a constraint in a subsystem derived from a system specified in the supplier); and *copy*, when the text in the client is a read-only copy of the text in the supplier.

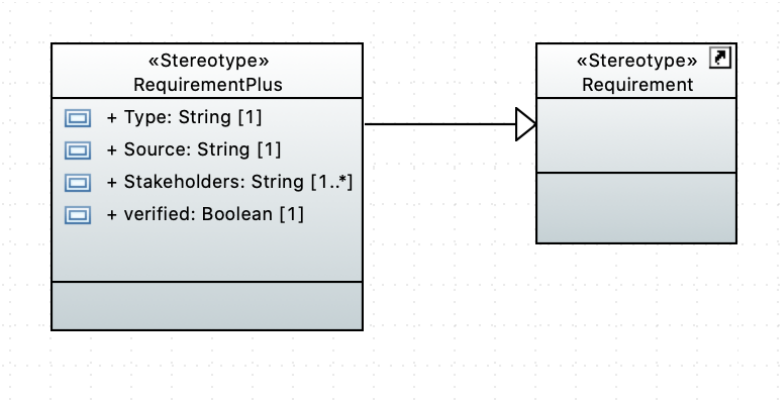


Figure 3: Requirements profile.

Finally, all the requirements have been organised in different packages that represent their type, so it is easier to access the diagram. Moreover, for the MDAO connection, the requirements that will be fulfill by the MDAO process are more related, obviously, to the design and performance package, which is decomposed in three other packages (Mission, landing and take-off performance). SysML also generates automatically requirement tables with all the refined information.

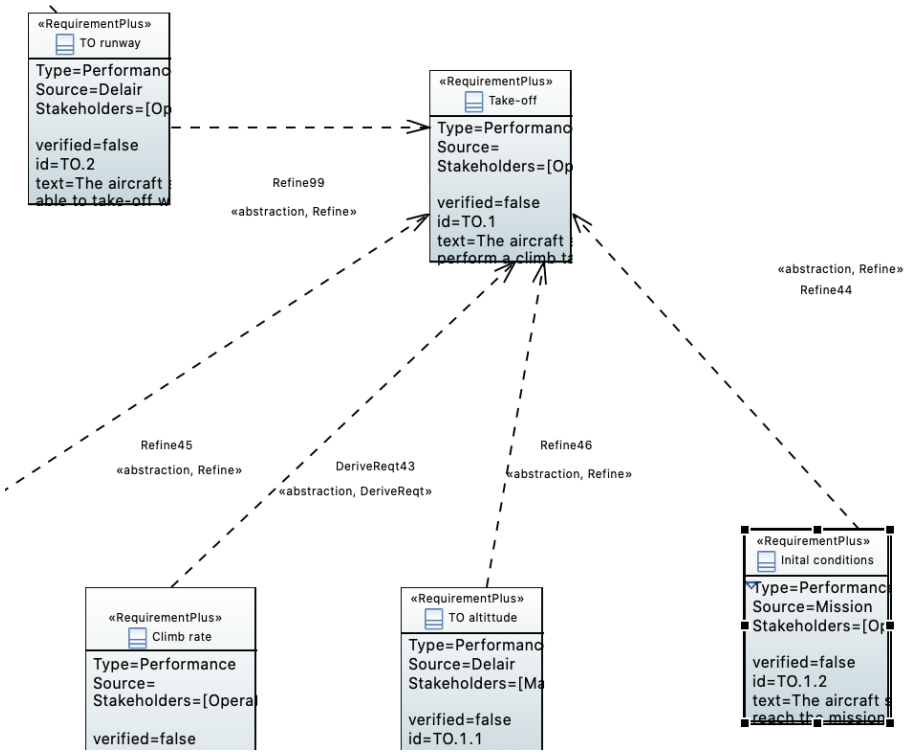


Figure 4: Requirements diagram with refine and derive connections.

Requirement connection with MDAO

Defining a connection between the requirements and the MDAO model is the main reason for starting this research process and, thereby, it is the last objective to accomplish. The great interest in this connection is due to the reduction in the time effort of the design process as, nowadays, there is no optimized process to link the requirements that may be take into account or be satisfied in the analysis and optimization design. Thus, these parts are performed independently and a connection will lead to a more efficient process.

Although this connection can only been established once the MDAO model has been implemented in Papyrus, it is important to highlight in this section that the Requirement diagram has a type of link called *satisfied*, which refers to a block defined in the BDD (Block Definition Diagram) from SysML, that is identified as the party responsible for fulfilling the requirement. As it will be explained later, the MDAO model will be reproduced in the MBSE using this BDD diagram. By doing it, the disciplines from MDAO are defined as blocks, which can be linked with those requirements they will be in charge of by using the *satisfied* connection. In this way, it is possible to fulfill all the design and performance requirements identified prior to the design process. An example can be seen in Fig. 5, where the discipline Mission satisfies the cruise speed limit that have been taken into account in the MDAO model.

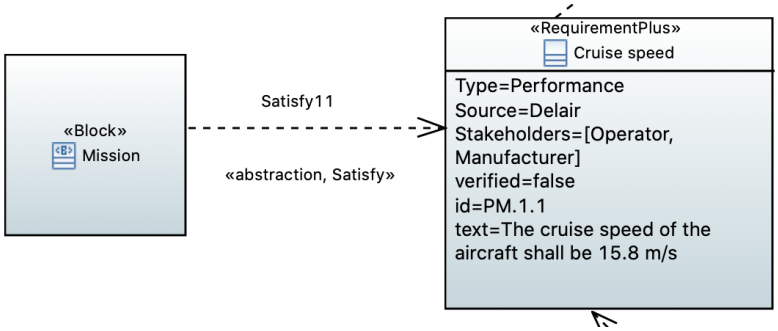


Figure 5: Satisfy connection between block and requirement.

3.2 MDAO implementation on Eclipse Papyrus

The MDAO representation on MBSE was thought to be necessary because it would allow the developed MDAO to exist right inside MBSE model. This allows the previously developed requirements diagram to be directly connected to the (representation of) MDAO model. To represent the MDAO model on Papyrus (MBSE platform), BDD was used. As Block is a representation of Discipline, Property (Attribute) within each Block represents variables. Furthermore, Port and ItemFlow were used to represent the variables exchange among disciplines. However, the basic BDD of SysML alone is not capable of wholly representing the MDAO model. Given that, a profile was created to further extend the BDD.

Profile definition

Initially, the meta-class Class is extended by a stereotype called Discipline as seen in the top left of Fig. 6. This is to have each discipline in the MDAO model to exist in the BDD.

Secondly, the meta-class Property of a Block is extended by a stereotype called "DisciplineVariable" as seen in the middle of Fig. 6. The DisciplineVariable stereotype has two attributes to better represent the MDAO variables. The two attributes are "Description" and "Unit". While "Description" was thought to be used for referencing purpose, "Unit" will directly represent each variable's unit. With this extension, each variable can be represented with a Block's Property.

Thirdly, the meta-class Port is extended by a stereotype called "Artefact". This extension of Port has two attributes: "direction" and "VariableTransfered". The "direction" attribute has a specially defined Enumeration Type called "Direction" to specify whether the Port receives (input) or sends (output) DisciplineVariable. The VariableTransfered specifies which DisciplineVariable going through a Port. Note that the "Association" relation was used to specify that the VariableTransfered attribute refers to the DisciplineVariable stereotype. With this extension, the variables exchange in the MDAO model can be represented.

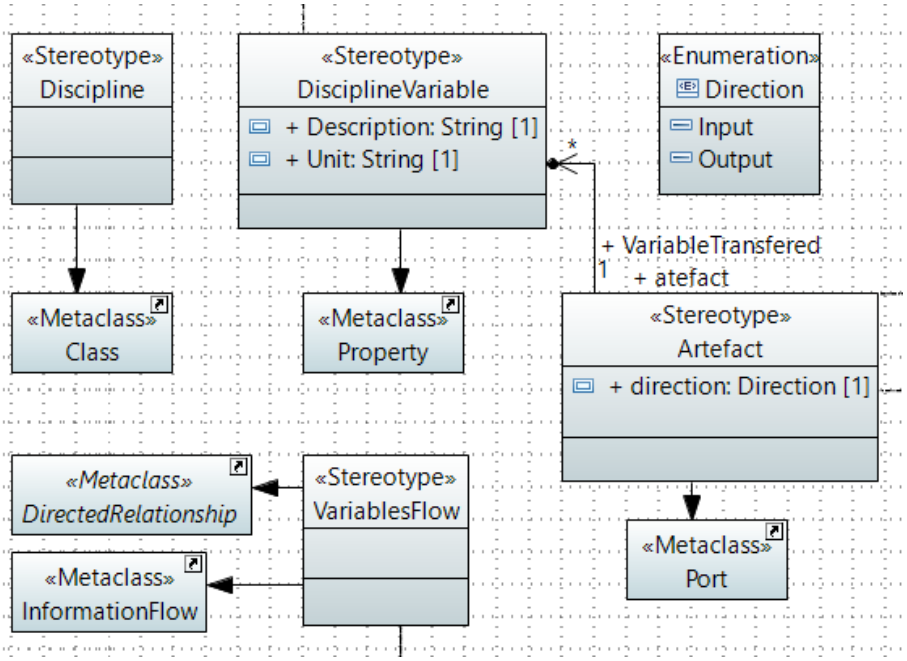


Figure 6: Profile diagram.

Lastly, the meta-class DirectedRelationship and InformationFlow were extended by a stereotype called VariablesFlow. Basically, this was done so that the ItemFlow entity in the BDD can have more than one variable and to accommodate some constraints as it is mentioned in the following section.

OCL constraint language

As it is defined in the OMG documentation [13], Object Constraint Language (OCL) is a formal language used to describe expressions on UML models and, thus, it can be extended to SysML models (as in Papyrus SysML packages stereotypes UML meta-classes). The OCL constraints can be applied to any kind of meta-class, Stereotype, operation, guard, message, action or attribute, which will be the Context of the constraint defined [14], [15].

OCL can be used for several purposes, but for this project the common use is for specifying invariants for the Stereotypes created in the profile and for specifying target (sets) for messages. Whenever a specific OCL constraint is defined, it imposes a boolean condition stated by the invariant contained in it and affects only the Context. Papyrus incorporates a tool for performing the OCL validation and if any condition is not satisfied, this validation will give an error.

The main idea is to use OCL as a validation method to implement the MDAO models to Papyrus. As it is explained later, the BDD diagram used to duplicate the MDAO model in MBSE can use the profile previously defined and, consequently, any OCL constraint contained in it. Therefore, any MDAO model can be implemented in Papyrus and use this profile and, moreover, any modification on an existing one will not affect the OCL validation (as long as all the invariants are still satisfied). This provides a versatile profile capable of being adapted to any model and with a validation process already incorporated.

The OCL constraints added to the profile are used to validate the BDD diagram consistency with the MDAO model and, additionally, they provide a guide for building this diagram that will be of help to establish the connection between MBSE and MDAO when using XML. Six OCL constraints were added to the model, the three first ones have *Artefact* as Context and the last three, *VariablesFlow*:

- *PortType*: verifies that the port is typed with the classifier (the *Discipline* block) of the variables contained in it.
- *PortVariableName* (represented in Fig. 7 and *PortVariableType*: check that the variables contained in a port of a Discipline are also contained in the Discipline with the same name and type.
- *FlowInput*: used to verify that the set of variables in a *VariableFlow* sent between ports is exactly the same as the set of variables in the Input port.
- *FlowOutput*: check that the set of variables in a *VariableFlow* is included in the set of variables of the Output port. The reason for not declaring same sets as done with the Input port is that all the variables sent from a Discipline, which may share variables with more than one Discipline, are contained in the same Output port.
- *InputOutput*: similar to the previous one, but between the Input and the Output port connected with a *VariableFlow*.

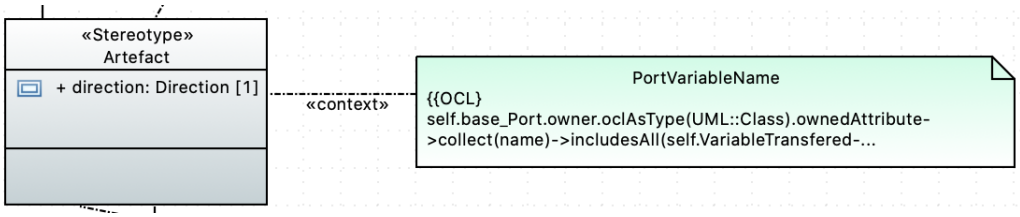


Figure 7: OCL constraint (*PortVariableName*).

All these constraints will be taken into account for building the BDD diagram and the XML file that will be explained in the last part of the investigation methods. Nevertheless, more constraints can be added to improve the profile and the validation process,

or even constraints coming from the requirements for the computation of the value of some discipline variables (e.g. a limit value that cannot be exceeded for a special kind of variable). It is necessary to remark that Papyrus gave some problems when trying to define the OCL constraints in the SysML diagrams. Consequently, for the last example, the constraint should be apply on the discipline variable stereotype from the profile and referring to its default value, such as follows:

```
if self.base_Property.defaultValue.oclIsTypeOf(UML::LiteralReal) then
self.base_Property.defaultValue.oclAsType(UML::LiteralReal).value<20.0
else
true
endif
```

Listing 1: OCL constraint for discipline variable value

The constraint in List 1 is applied to every discipline variable whose default value has been defined as a LiteralReal and checks that the value given is lower than 20. This is a general example, but there are several ways to delimit the number of variables to which the constraint is applied.

Finally, although it is not really necessary for the model, it would be useful to add an OCL constraint in the requirement stereotype from the requirement profile in order to verify that all the design and performance requirements are satisfied by at least one Discipline block from the BDD. However, in this project some problems with this process in Papyrus made impossible to perform the validation.

Block Definition Diagram

Once the profile with the OCL constraints is completely defined, it is possible to build the Block Definition Diagram (BDD), which is going to be the representation of the MDAO model in Papyrus (MBSE model). However, so far in this section, this implementation is done independently, which means that the BDD is totally built in Papyrus tool, just using a specific MDAO model as reference in order to add all the Discipline blocks, its discipline variables and all the ports and connections between Disciplines.

First of all, it is necessary to import the profile defined so that all the stereotypes can be applied. Once imported, the following elements have to be stereotyped: the block (with *Discipline* stereotype), its properties or attributes (with *DisciplineVariable*), its ports (with *Artefact*) and the itemflows (with *VariablesFlow*), which represents the flow of variables between ports.

Figure 8 displays in red circles all these stereotypes in the Discipline block called Mission from our MDAO model. This discipline used the variables vit_eco and vit_max from the Driver (thus, there is a port called *Input mission-driver*) to compute the duration of the cruise flight (duree_crois). Then, it send the variable computed in it, duree_crois, and vit_max through the port *output mission* and these variables flow through the *Flow miss-mass* to other disciplines. Finally, it can be seen also that the stereotype *Artefact* from the *output mission* port contains these variables in the attribute *VariablesTransfered*.

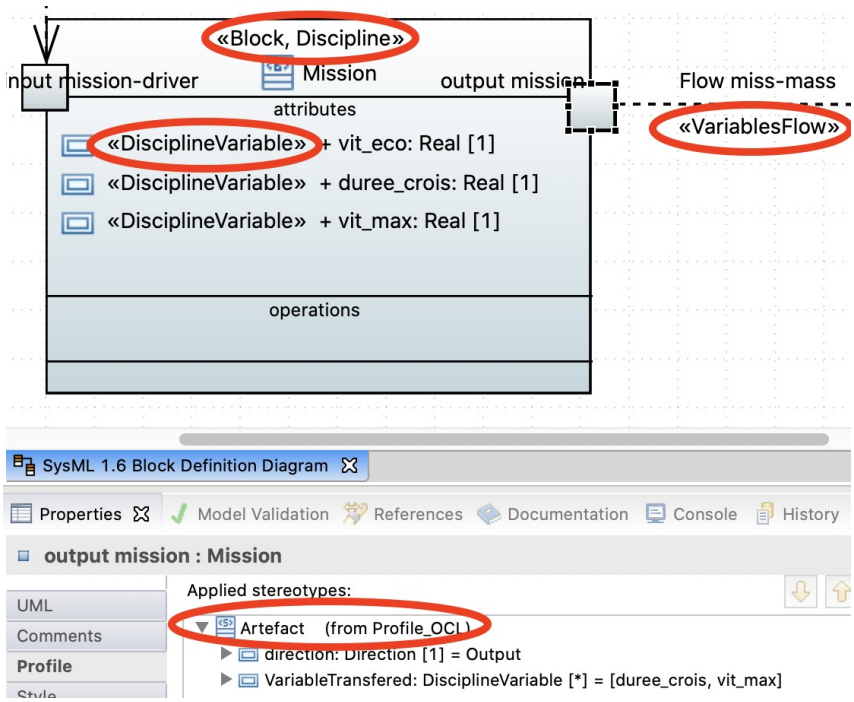


Figure 8: Example of Block in BDD with all the stereotypes for MDAO model.

By repeating the stereotype process as it has been done in Fig. 8, it is possible to implement an entire MDAO model in MBSE. Despite the fact that this could be already understood as a connection between the models, the process is not optimized. As it has been said, the models are built independently (the BDD or the MDAO are both built from scratch) and any improvement, where more elements have to be introduced, lead to a different change in each model. This is the reason for searching a real connection that allows to create the BDD and the OpenMDAO at the same time, reducing the time effort and improving the validation process.

3.3 MDAO-MBSE connection

This connection will be useful as an iterative process for going from one model to the other. In other words, when a change has to be introduced in one model (for example, a new discipline block in the BDD is needed to satisfy a new requirement or new variables are added in a MDAO discipline to optimize the computation value of an objective function), the other connected model added the new elements at the same time.

In order to achieve this connection, it was found that when building a Papyrus project, this tool creates an XML file containing all the information of all the elements from the model with their applied stereotypes. This file is not a read-only file, but it can be overwritten and the changes will appear in the model and diagrams as well. Therefore, the model can be changed or even created in this XML file. Furthermore, Python has two libraries which allow to parse XML files and modify them by adding new elements, changing the existing ones or eliminating them. As OpenMDAO is coded in Python, XML offers an interesting opportunity to establish the desired link.

XML

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. A further documentation can be found in [16], but, basically, XML documents are a string of characters, composed of elements and subelements. Each element or subelement begins with ¡ and ends with ¿ and is defined by a tag and its attribute. The tag can be seen as the name given for the element and the attribute consists of a set of name–value pairs that can be used to characterised the tag.

An example is given below in List 2 and represents the discipline variable vit_max from Fig. 8. This is a subelement of the element that defines the block Mission. Its tag is *ownedAttribute*, which means that it is an attribute owned by a package element (in this case the block Mission) and the attribute is the rest of the line contained between ¡ and ¿. In the attribute there are three name-value pairs: the first one specifies that this subelement type is a Property of the block; the second one is an id; and the third one is the name given to this owned attribute and displayed in the model. The second line is a subelement of the this *ownedAttribute*. It represents the value of the variable, which is going to be 15.8 m/s. Finally, the last line is the stereotype *DisciplineVariable*.

```
<ownedAttribute xmi:type="uml:Property" xmi:id="TJL_xcP" name="vit_max">
    <defaultValue xmi:type="uml:LiteralReal" xmi:id="_5x4g" value="15.8"/>
</ownedAttribute>
<Profile_OCL:DisciplineVariable xmi:id="_zWCbD8" base_Property="TJL_xcP"/>
```

Listing 2: XML sample for vit_max variable

Just by observing in a same way as in List 2, the different elements in an XML file from an already built BDD, it is possible to identify all the discipline blocks, the discipline variables, the ports and the itemflows. It is easy to conclude that, knowing all the tags and attributes for each element of the diagram, it is only necessary to learn how to access or create those elements from an XML file using Python and, thereby, in the OpenMDAO model.

Luckily, Python has two libraries that parse XML files and allow to access the elements and subelements and, moreover, to create them. These libraries are called *minidom* and *ElementTree*.

Both libraries have similar features and functions to modify XML files. An interesting basic guide that explains them can be found in [17] and [18]. However, it seems that *minidom* is better to access existing elements and *ElementTree* eases the creation of new ones. This remark is really important because depending on which model should copy a new element added to the other model, one library or the other will be used.

When a new element is introduced in the BDD diagram from the MBSE model, the OpenMDAO model will have to add it as well. For this case, the element is automatically added to the XML file from the diagram. Hence, the OpenMDAO will just have to read it from the XML, so *minidom* is more comfortable to access the element and its attributes. For example, if a new discipline variable is added to a Discipline block, the OpenMDAO model will use its name and its default value attributes to specify the independent variable inside the discipline, its value and the connection, if necessary, with the variable in other disciplines.

For the other case where a new element or elements are introduced in the MDAO and, straightaway, to the BDD, they will be added manually in the Python code with the

OpenMDAO model. For that purpose, *ElementTree* helps better to create the elements or subelements. It is essential to know which type of element is being upload to the XML file since the tag and attributes are defined in a different way. Looking at the XML sample described in List 2, it could have been created after the variable vit_max was defined in the OpenMDAO with a value of 15.8. Once the XML file is updated in Python, it can be imported in Papyrus and the variable will appear in the BDD diagram with the stereotype and the Default value as it can be seen in Fig. 9:

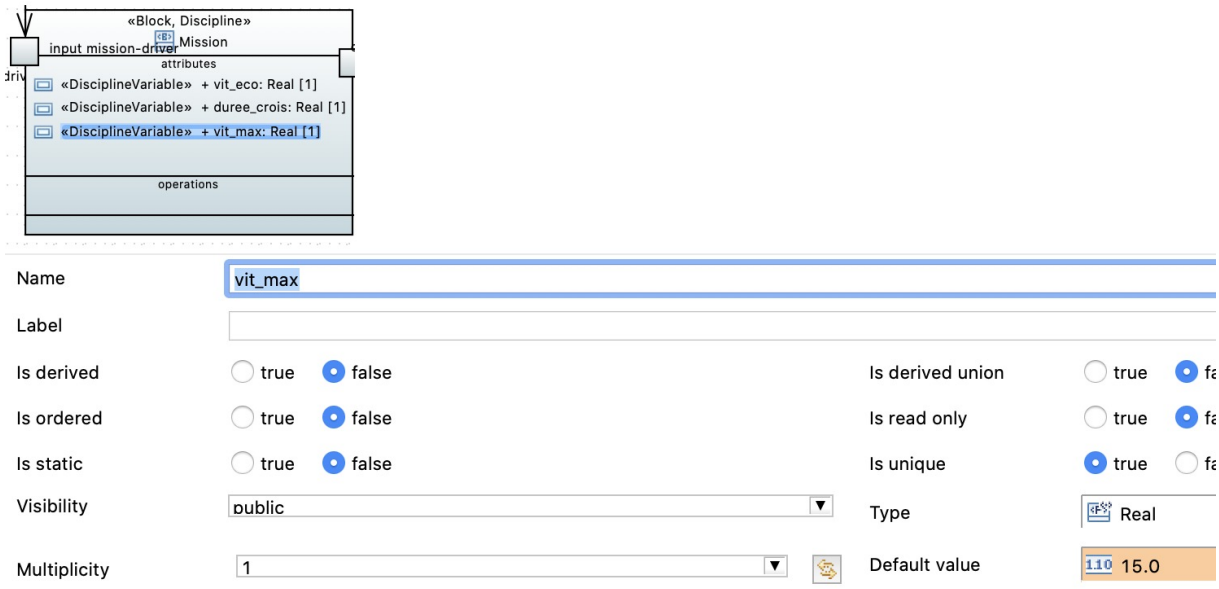


Figure 9: vit_max introduced to BDD using XML.

Both process with all their steps are explained in detail in [12]. Still, some of the functions from the libraries used need to be highlighted:

- *parse*: present in both libraries, this function is mandatory to be able to parse and modify the XML file.
- *getElementsByTagName()* and *getAttribute()*: only from *minidom*, the first function finds and creates a list of all the elements or subelements of the file that have the same tag name as the one defined in the brackets. The second function is used after the first one to access the value of the attribute name specified in brackets from one of the elements in the list previously built.
- *SubElement(parent, tag, attrib=)*: from *ElementTree*, this function creates an element instance, and appends it to an existing element (*parent*). *tag* is the subelement name and *attrib* is an optional dictionary, containing element attributes with their values. An example of how to create the XML lines in List 2 with this function is shown in List 3. In this list, firstly, it can be seen that the variable vit_max gets the value 27.8 in the MDAO model. Then three subelements are created: the first one specifies the *ownedAttribute* of the parent element, the Discipline block *Mission*; the second one is the applied stereotype *DisciplineVariable* from our Profile_OCL and is appended to App which is the first element of the XML and represents all the namespaces of the model; the third one is the default value (27.8) whose parent is, obviously, the first subelement created.

- *write("Output.xml", xml_declaration=True, encoding="UTF-8", method="xml")*: from *ElementTree*, writes the element tree to a XML output file with a UTF-8 output encoding. This XML file is the one with all the desirable modifications that will be imported to the Papyrus diagram.

```
top_1['p.vit_max']= 27.8
vitmax=str(top_1['p.vit_max']);
vitm=ET.SubElement(miss, 'ownedAttribute', attrib={'xmi:id' : vite_id, 'name' : "vit_max"
    , 'xmi:type' : "uml:Property"});
ET.SubElement(App, 'Profile_OCL:DisciplineVariable', attrib={'base_Property' : vite_id, '
    xmi:id' : vite_id2});
ET.SubElement(vitm, 'defaultValue', attrib={'xmi:id' : vite_value_id, 'xmi:type' : "uml:
    LiteralReal" , 'value' : vitmax});
```

Listing 3: Example code for creating the XML code of List 2 in Python

Taking into account all this information, it is possible to conclude that a link between model can be established. As it was explained before, it could be done by three different ways:

- Building firstly the MDAO model and, at the same time, building the XML in Python manually with *ElementTree* so that it will be imported to the MBSE model for the validation of the model itself and the requirements.
- Other way could be creating first a BDD diagram to see which block satisfies each requirement, so that the XML is automatically built and, next, accessing all the elements from that file with *minidom* for building the MDAO model to perform the analysis and optimization.
- Additionally, when both models have already been implemented, it is easier to change anything in one model and incorporate it to the other one without affecting the validation process.

In order to see the steps of these link processes, take a look at [12]. A final remark has to be done to explain the general structure of the XML file that Papyrus uses and is able to read and import. The principal element of this file is the group of namespaces used. In XML it is mandatory to have this namespaces registered every time the *ElementTree* library is used with the function *register_namespace()* as it is done in List 4. Furthermore, to be able to write correctly the namespaces in the output file, every namespace has to be referred with the second name given in the function *register_namespace*.

```
ET.register_namespace('xmi',"http://www.omg.org/spec/XMI/20131001");
ET.register_namespace('Blocks',"http://www.eclipse.org/papyrus/sysml/1.6/SysML/Blocks");
ET.register_namespace('Profile_OCL',"http://www.isae.fr/certification/1");
ET.register_namespace('uml',"http://www.eclipse.org/uml2/5.0.0/UML");
```

Listing 4: Python code for registering the namespaces

Therefore, the code in List 3 will not work correctly. It was displayed like that form to keep the same format of the previous XML in List 2. However, all the namespaces have to be changed. E.g. the *'xmi:id'* will be changed to the name given in *register_namespace*, *'http://www.omg.org/spec/XMI/20131001id'*, and the third line will be written like this:

```
vitm=ET.SubElement(miss, 'ownedAttribute', attrib={'{http://www.omg.org/spec/XMI
    /20131001}id' : vite_id, 'name' : "vit_max", '{http://www.omg.org/spec/XMI/20131001}
    type' : "uml:Property"});
```

Appended to the namespaces element (called App in Python), there are several subelements. The first one and most important, is the uml package, where all the elements of the diagram have to be appended to, and after this element, all the stereotypes applied. Hence, when stereotyping in Python, the parent of the stereotype tag is not the variable to which the stereotype is applied, but the first element App.

4 RESULT AND ANALYSIS

As it was explained in the final part of the previous section, there are several ways to connect the models and perform a validation. During the project, although all ways were tested to see their viability, the main methodology used was to create a XML file in Python from an existing MDAO model due to the fact that several models were already available from last year research study. Hence, this process can be described as follows:

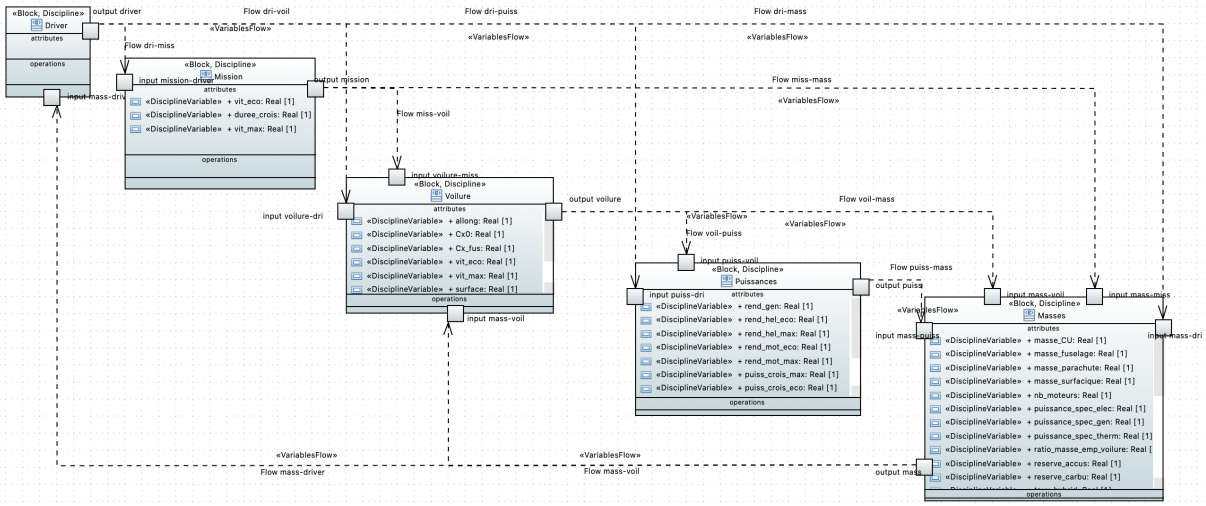


Figure 10: BDD result after importing the XML file created in MDAO.

1. In order to be able to import a correct XML file to Papyrus, the first step is to create an empty BDD in this tool and apply the *Profile_OCL* to use all their stereotypes. The XML file from this empty BDD will contain the first element with all the namespaces needed and the structure of the XML will be ready to be used in the MDAO model. It is important to highlight that to register the namespaces of the stereotyped profiles it is necessary to use them in one element of the empty BDD. Thus, it is necessary to create an empty block and stereotype it with *Discipline*.

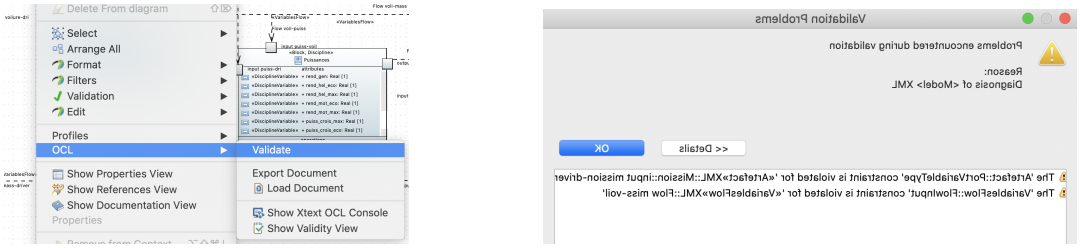
2. The MDAO model used was the one from the mock-up in Fig. 10. In order to see results not too long this model is perfect. All the Python code was written in a Jupiter Notebook containing the model. The procedure was, after parsing the XML file created before from the empty BDD, writing the different elements in the XML file at the same time the OpenMDAO defined them. Thus, firstly, all the Discipline blocks were created, including the *Driver* with all the independent variables. Then, after modelling each discipline in MDAO, all the ports, variables and connections of them were added to XML.

3. Now the XML is already finished and, consequently, the BDD is built. It only remains to import the XML file in Papyrus and the model in MBSE is practically

done. After importing the file, the model explorer will show that all the elements from MDAO have been added. However, the diagram console will not display them. This is not a problem because the elements exist in the model and the validation process of the OCL can be performed already. Just by dragging each element in the same order as in the XDSM of the MDAO model, it is possible to replicate it and the final result is shown in Fig. 10.

4. Afterwards, the OCL validation is performed to see the consistency of the model and to test any constraint that can be added to the profile to maybe fulfill the requirements in some variables or safety requirements. The validation in the BDD is really simple as by right clicking the diagram, there is a feature of OCL containing the validation bottom (see Fig. 11a). If any error is found in the model, it will be displayed in the model validation console as in Fig. 11b, along with the name of the OCL constraint violated and the variables that violate it. In Fig. 11b, it can be seen that two OCL were not satisfied correctly: the first one refers to *PortVariableType* and means that there is a variable in the port *input mission-driver* from the discipline *Mission* that has not the correct type; the second one refers to *FlowInput* and states that the itemFlow *Flow miss-voil* is sending a set of variables to an input port that does not contain that specific set.

5. Once the model has been validated, a final connection with the requirements can be done as in Fig. 5 to connect the Discipline blocks with the requirements. This is the last and most important step as it provides a connection between MDAO model and requirements.



(a) OCL validation feature in BDD. **(b) Model validation console with OCL violated.**

Figure 11: OCL validation

Comparing the BDD result in Fig. 10 with the initial MDAO model Fig. 1, it is possible to see the a lot of similarities between the models. The procedure leads to a good implementation of the MDAO in the BDD and the validation gave the good results expected.

It is necessary to highlight that for each discipline in the BDD, there is only one output port with all the variables that are sending from that discipline, regardless of the destiny of the variables.

Whereas it is not the same situation for the input ports, which only contain the variables sent through the *VariableFlow*, meaning that the origin of the variables has to be considered and, for each set of variables coming from a different discipline, one input port will be added. The reason is that, for the way the OCL language was defined for the *VariableFlow*, it checks that the set of variables in it and in the input port (*FlowInput*) has to be the same and, hence, no variables coming from other disciplines can be introduced.

This could be changed by modifying the OCL and use the function *includesAll* as it was done with the output ports, so only one input port will be needed. But for the better understanding of the model and because of the convenience building the XML in Python, it was decided to build the model in this way. The validation process is no affected anyway.

Finally, once the method was finished, it was time to check in that all the design and performance requirements were satisfied by, at least, one discipline block looking at how the MDAO model was built and connecting them as in step 5). Then, it was perceived that some requirements were still not connected (for example one referring to a turn maneuver). That is why in Fig. 2, the discipline *Turn* was added. This step was not finished during the time of the research, but an idea of how to create the discipline can be concluded by looking at how it was explained the implementation of the XML in the MDAO using the *minidom* library.

5 CONCLUSION AND PERSPECTIVES

This link has been a first attempt to connect MDAO and MBSE and further investigation is needed to improve the efficiency of the connection, especially in terms of time effort (still, it is complicated and long to build the XML file in Python and there is no much time saved comparing with building the models independently). However, it is a good start and the validation process seems to work as expected.

More than one MDAO model was tested using this link and good results were obtained. In terms of effectiveness, if no model has been built, it is more pragmatic starting with the MDAO model in spite of the fact that the XML file has to be written manually. The main reason is that, if the BDD is built first, there is not much time saved reading the XML file with Python and the MDAO can be developed independently without using much more effort. However, if at the same time you are building the MDAO and the XML, the BDD will be automatically developed. Furthermore, [12] gives a guide to write the code for the XML and this process can be automated.

Even if the step 5) in the results section 4 provides a first link with requirements, the number and type of OCL constraints added to the model for a better connection to the MDAO model and the requirements is up to the user of this procedure. The profiles are available to be modified and add them, and a great variety of possibilities is offered. One example was given in List 1 to limit the value of a set of variables.

To conclude this paper some perspectives can be mentioned in order to keep developing these models and their connection:

1. Improve MDAO workflow especially on Mission accuracy (*Take-off*, *Landing* and *Climb*) and finalize *Turn* discpline.

2. Build other SysML diagrams for the MBSE model (Use case diagram and physical BDD were already developed, but there are other diagrams that may help to understand the model).

3. Investigate in the possibility of adding the OCL constraints directly in the BDD so that their context would be a specific discipline variable.

4. Make an OCL constraint work in the Requirements diagram so that the validation process takes into account that all requirements are satisfied by one or more discipline blocks. During the project the requirements profile was giving unexpected

error when trying to add any kind of OCL constraint. The idea is to introduce the constraint given below:

```
self.satisfiedBy->notEmpty()
```

5. As it was explained in [8] and [9], finding a link with MBSA model is a really interesting research for many companies these days.

ACKNOWLEDGEMENTS

This work was supported by the Defense Innovation Agency (AID) of the French Ministry of Defense (research project CONCORDE N° 2019 65 0090004707501). This work is part of the activities of ONERA - ISAE - ENAC joint research group.

REFERENCES

[1] P. Choy and B. Danet. AVOCETTES UAV conceptual design tool. Technical report, ONERA, 2020.

[2] J. S. Gray, J. T. Hwang, J. R. Martins, K. T. Moore, and B. A. Naylor. Openmdao: An open-source framework for multidisciplinary design, analysis, and optimization. *Structural and Multidisciplinary Optimization*, 59(4):1075–1104, 2019.

[3] R. Lafage, S. Defoort, and T. Lefebvre. Whatsopt: a web application for multidisciplinary design analysis and optimization. In *AIAA Aviation 2019 Forum*, page 2990, 2019.

[4] A. B. Lambe and J. R. Martins. Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes. *Structural and Multidisciplinary Optimization*, 46(2):273–284, 2012.

[5] M. Hause et al. The sysml modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9, pages 1–12, 2006.

[6] L. Delligatti. *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley, 2013.

[7] `https://delair.aero/delair-commercial-drones/dt26x-surveillance/`, 2020.

[8] K. Thanissaranon. Global model of aircraft design: from performance requirements towards architectures optimization. 2020.

[9] C. G. Rubio. Global model of aircraft design: from performance requirements towards architectures optimization. 2020.

[10] A. Essari. *Estimation of component design weights in conceptual design phase for tactical UAVs*. PhD thesis, Univerzitet u Beogradu-Mašinski fakultet, 2015.

[11] D. Raymer. *Aircraft design: a conceptual approach*. American Institute of Aeronautics and Astronautics, Inc., 2012.

[12] C. G. Rubio and K. Thanissaranon. Supplementary document for global model of aircraft design: From performance requirements towards architectures optimization. 2021.

[13] Object constraint language. https://www.omg.org/spec/OCL/2.4/PDF, 2014.

[14] Acceleo/ocl operations reference. https://wiki.eclipse.org/Acceleo/OCL_Operations_Reference, 2019.

[15] L. Audibert. *UML 2: De l'apprentissage à la pratique*, volume 298. Ellipses, 2009.

[16] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, et al. Extensible markup language (xml) 1.0, 2000.

[17] The elementtree xml api. https://docs.python.org/3/library/xml.etree.elementtree.html, 2021.

[18] S. Robinson. Reading and writing xml files in python. https://www.omg.org/spec/OCL/2.4/PDF, 2021.