NASA/CR-20250007048

Model-Based Systems Analysis and Engineering: The Development of Enhanced Model-Based Systems Analysis (MBSA) and Model-Based Systems Engineering (MBSE) Couplings for Practical Applications

Mingxuan Shi
The Boeing Company, Everett, Washington

Paul Mokotoff and Gokcin Cinar
University of Michigan, Ann Arbor, Michigan

August 2025

NASA/CR-20250007048

Model-Based Systems Analysis and Engineering: The Development of Enhanced Model-Based Systems Analysis (MBSA) and Model-Based Systems Engineering (MBSE) Couplings for Practical Applications

Mingxuan Shi
The Boeing Company, Everett, Washington

Paul Mokotoff and Gokcin Cinar
University of Michigan, Ann Arbor, Michigan

Prepared under Contract 80GRC023CA045

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

August 2025

Acknowledgments

This project and report reflect the combined efforts of the Boeing MBSA&E team. Funding for this effort was provided by the Sustainable Flight National Partnership Program, through the Advanced Air Transport Technology Project. The work was performed under NASA Contract 80GRC023CA045. Eric Hendricks was the NASA MBSA&E Program Manager. Ty Marien was the Contract Technical Representative. Bijan Fazal was the NASA sub-task lead for this work. From NASA, Bijan Fazal and Irian Ordaz provided support on the use of the MBSA&E tools. At Boeing, Ranald Engelbeck, Sean Wakayama, Juan Mendo, Luis Garcia-Mozos, and Alex Carrere supported work with the MBSA&E MBSE tasks. Melinda Refford from Collins supported the creation of the sysML models for sharing demonstration.

This work was sponsored by the Advanced Air Vehicles Program at the NASA Glenn Research Center.

Level of Review: This material has been technically reviewed by expert reviewer(s).

Abstract

The National Aeronautics & Space Administration (NASA) Model-Based Systems Analysis & Engineering (MBSA&E) effort developed a toolset for system-level vision-vehicle performance and technology assessments in support of the Sustainable Flight National Partnership (SFNP). NASA sponsored a set of projects to stimulate innovation and develop industry partnerships supporting the MBSA&E effort. The Development of Enhanced Model-Based Systems Analysis (MBSA) and Model-Based Systems Engineering (MBSE) Couplings for Practical Applications project was one of six projects performed by Boeing. This project developed the metamodel and sysML profile to support the connectivity between the MBSA and MBSE. An airplane systems engineering model has been created following the metamodel and profile, which include the requirements, architecture, and performance of an airplane. This system engineering model was also shown to support the model sharing with the industrial partner Collins Aerospace. Besides, Dr. Gokcin Cinar's team in University of Michigan developed the software that implemented the connectivity between the MBSE and MBSA. Finally, it is concluded that this project developed the necessary infrastructure to support the digital connectivity between the MBSA and MBSE sides.

Contents

Introduction

Background and Objectives

Under the National Aeronautics and Space Administration (NASA) Model-Based Systems Analysis & Engineering (MBSA&E) Program Phase I, the goal is to develop the overall framework and assessing its performance on a set of example aircraft, which integrates the disciplinary analysis tool set with the systems engineering artifacts. Accordingly, the main objective of CLIN 003 is to couple the OpenMDAO-based framework for Model-based Systems Analysis (MBSA) to Model-Based Systems Engineering (MBSE) tools.

The main tasks of CLIN 003 are to: 1. Develop a metamodel that the description systems engineering model to follow so that the systems engineering model can be properly integrated with the disciplinary analysis toolset; 2. Develop a descriptive systems engineering model with a profile that follows the developed metamodel in the previous task; and 3. Integrated the systems engineering model with the disciplinary analysis. An overarching diagram is presented in to show the interaction between different CLINs in Figure 1. As shown in this figure, for CLIN 003 specifically, the systems engineering artifacts include a MBSA&E metamodel, a MBSA&E profile, a descriptive system model, and a MBSA definition pulled from the disciplinary analysis. In addition, the MBSE-MBSA coupling connector was developed under CLIN 003 by the academic partner from University of Michigan, Dr. Gokcin Cinar's research group. Regarding the interactions of CLINs, CLIN 001 informs the systems engineering modeling about the data hierarchy to be implemented from system level to component level, including the data structure of the value properties; CLIN 004 and CLIN 005 handle the model sharing task which includes the sharing of Boeing's systems engineering model to the industrial partner Collins and the sharing of Collins' systems engineering model back with Boeing, including a Comprehensive Model Sharing approach and a Narrow Model Sharing approach.

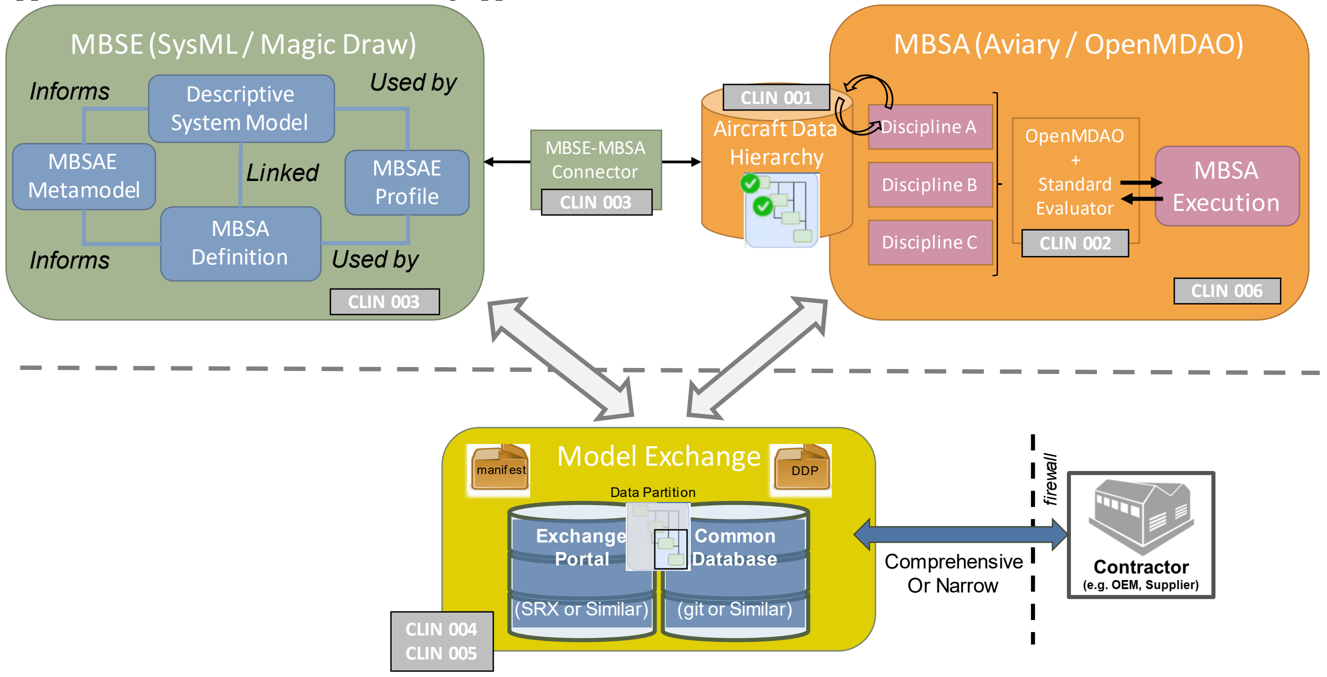


Figure 1 Interaction of Different CLINs under NASA MBSA&E Program Phase I

This report summarizes the methodologies that the MBSE utilizes to develop the systems engineering artifacts, as well as the systems engineering models themselves. This report also discusses the software development to support the digital MBSE-MBSA connectivity. Please note that the best practices about creating systems engineering models or implementing MBSE-MBSA digital connectivity is not included in this report. Such best practices can be found in other deliverables.

This report is organized as follows: the scope of the systems engineering modeling and MBSE-MBSA digital connectivity will be discussed first; then the development of the Metamodel and sysML profile will follows;

then the airplane model will be presented with the details how the Metamodel and profiles are followed; next, the sysML model will be shown to support the model sharing with the industrial partner; then the software development will be discussed; at the end, the conclusions will be drawn.

Modeling and Software Development Environment

For CLIN 003, the systems engineering modeling environment is MagicDraw 2022x Refresh 1 which is a tool to perform MBSE modeling and analysis tasks, and the systems modeling syntax is Systems Modeling Language (sysML) version 1. It should be noted that most of the modeling methodologies are tools and syntax agnostic, but the examples included in this report are based on MagicDraw 2022x R1 and sysML v1. The MBSE-MBSA digital connectivity software were developed using Jython which is an implementations of Python language in Java environment.

Scope of the Systems Engineering Modeling and MBSE-MBSA Coupling

The most important thing to prepare for systems engineering model to understand the scope of model, that is, the purpose of the model and how to use the model. It is recognized that a systems engineering model can easily become extremely complicated and overwhelming for human to track all aspects of the systems, because the systems nowadays themselves become more and more complicated, considering the increasing number of the system layers, requirements, regulations, integrated functionalities, etc. Therefore, the rule of thumb here is that one item should be modeled in the systems model if and only this item is necessary to fully the model's purpose and its use case. This means that anything that will not be used should not be modeled at all. The fundamental reason is that a model is not useful if it cannot be fully comprehended.

Purpose and Usage of the Systems Engineering Model

There are many purposes or usages for systems models, which include and not limited to:

- Derive requirements from the top-level requirement to ensure correct requirement parent-child relations
- Development the validation and verification relations between the requirements and corresponding artifacts
- Generate Development Assurance (DA) artifacts (safety analysis, regulation compliance document, etc.) based on airplane functional and logical models
- Share information between airplane integrator and suppliers
    - Communicate requirements
    - Communicate interface compatibility
    - Communicate component/subsystem integration analysis
- Establish the traceability between the design decisions and the disciplinary analysis results
- Validate or verify the requirements based on test/experimental/simulation results

For each of the purposes or usages, the needed modeling aspects are different. For example, if the purpose is to generate the DA document used for airplane and system-level safety analysis, the functional, logical, and requirement models are needed, but the interface is not needed. If the usage is to share interface with the suppliers, then the functional model and requirement model may not be necessary.
In this work, the purpose of the descriptive systems engineering model can be summarized as follows:

1. Support the communication with the industrial partner Collins Aerospace on the requirements and interface between the Environmental Control System (ECS) and its component Air Cycle Machine (ACM)
2. Establish the traceability between the airplane design decisions and the disciplinary analysis
3. Support the verification of the airplane requirements through disciplinary analysis.

More details are discussed in the following subsections.

Communication with Suppliers/Collaboration Partners

One of the use cases in this work is to communicate the systems engineering model with the industrial partner Collins Aerospace on the interface, requirements, and the performance metrics associated with the supplier's part. It is well recognized that the airplane designer or integrator can only develop the airplane system to certain level, and after this level communications need to happen between the integrator and suppliers. For example, in this work the airplane can only define the define the system from the airplane level down to the ECS level which is comprised of ACM and an air distribution system. However, the ACM is not designed or manufactured by the airplane integrator but designed by one of the suppliers Collins Aerospace. Therefore, the requirements on interface and performance of the ACM need to be communicated with the supplier to have a valid design. And after the design is finalized, the design needs to be communicated back to the airplane integrator, so that the corresponding ECS requirements can be verified as the integration process starts from the component level to the top airplane level. It should be noted that only the scope is discussed in this section, while more details on the development of the model for sharing are provided in sections.

Establishment of Traceability between System Design Decisions and Disciplinary Analysis

Another important objective of this NASA MBSA&E effort is to create an integrated framework that can couple the systems engineering artifacts with the disciplinary analysis, so that the designers in the future can easily track the reasons why certain design designs are made, and then the designers can further evaluate if changes of design are desired or not. To accomplish this objective, the traceability between the design decisions and the disciplinary analysis is needed. In conventional sysML there are no existing elements to hold such information, therefore, a new metamodel is needed, which will be discussed in the following sections.

Verification of Airplane Requirements

The third objective of this systems engineering modeling effort is to demonstrate the requirements developed in the MBSE framework can be verified by simulation or disciplinary analysis. An example showing the verification of airplane requirements is illustrated in the Section Requirement Verification Demonstration.

To realize the previously presented two objectives, there are a few necessary items needed:

1. On the systems engineering side, there must be places to host the analysis data
2. The data structure and architecture of systems engineering model must be compatible with the disciplinary analysis numerical models
3. A connector between the MBSE and the MBSA so that the two sides can communicate the data and requirements with each other. It should be noted that this connector is being developed in the MagicDraw environment using the programming language Jython.

Scope of MBSE-MBSA Connectivity Development

It should be noted that the data from MBSE is not directly consumed by the disciplinary analysis programs, nor the data from disciplinary analysis is directly ingested into MBSE to instantiate the systems engineering model. However, the data exchange between the MBSE and MBSA happens at their interface, the ADH file. This ADH file serves at the interface of both the MBSE and MBSA environment. And the scope in CLIN 3 for the MBSE-MBSA connectivity is to develop software to support ADH file as the interface of the MBSE environment. The development to support ADH file as the interface of the MBSA environment is within the scope of CLIN 1, the ADH development. A brief introduction of the ADH file is also included here, while more details can be found in the report of CLIN 1. An ADH file contains a hierarchical data structure that represents a single airplane instance, including four major aspects: requirements, architecture, performance, and behavior.
In this sense, the scope of CLIN 3 for the MBSE-MBSA connectivity can be summarized as follows, considering the specific MBSE software environment MagicDraw 2022x R1:

1. The data contained in an ADH file shall be able to be imported to the sysML modeling environment to instantiate a systems engineering model, including requirements, architecture, and performance.
2. The data importing shall not affect the views created by systems engineers in MagicDraw 2022x R1.

3. Different ADH files with different values shall be able to be imported to MagicDraw 2022x R1 as different instances with different values.
4. The data imported from ADH file shall be able to support the requirement verification activity in MagicDraw 2022x R1, through instance tables or requirement tables.
5. The systems engineering models created in MagicDraw 2022x R1 shall be able to export to an ADH file following the ADH structure, if the systems engineering model follows the naming and structure conventions that are consistent with ADH structure.

Quantitative Analysis Framework and Scope

In this study, only conceptual design-level analysis will be performed, so the fidelity of the analysis will be at a relatively low level. The integrated MBSA-MBSE demo was conducted using an engine analysis in PyCycle[1], where the engine performance (e.g., thrust, fuel flow, TSFC, air mass flow rate, etc.) is assessed. More details can be found in the report of CLIN 001.

Metamodel development

Before the development of the descriptive systems engineering model, a metamodel is needed, and this section include the details of the development of the metamodel. The metamodel here refers to a model that describes what the basic elements in a systems modeling syntax are used to describe the systems and the relations among them. In this context sysML v1 is the syntax, and the basic elements and classes from sysML v1 are therefore used in the metamodel.

In this study, the use cases of the systems engineering model are as discussed in the previous section. Therefore, this model shall have: 1. A system architecture with logical elements with their corresponding value properties to host the data from MBSA; 2. A data hierarchy that is compatible with the disciplinary analysis; 3. Relations that link the design decisions and the analysis results; and 4. Elements that represent the information needed to communicate with the industrial partner. It should be also noted that this metamodel is compliment to the NASA's metamodel [1] to handle the coupling of MBSA and MBSE, and this metamodel is not supposed to replace the existing NASA's metamodel. The overall metamodel is presented in Figure 2, which include all the four aspects as discussed above. More details will be discussed in following subsections.

[1] PyCycle - An Cycle Modeling Tool For Design With Gradient Based Optimization, https://software.nasa.gov/software/LEW-19288-1

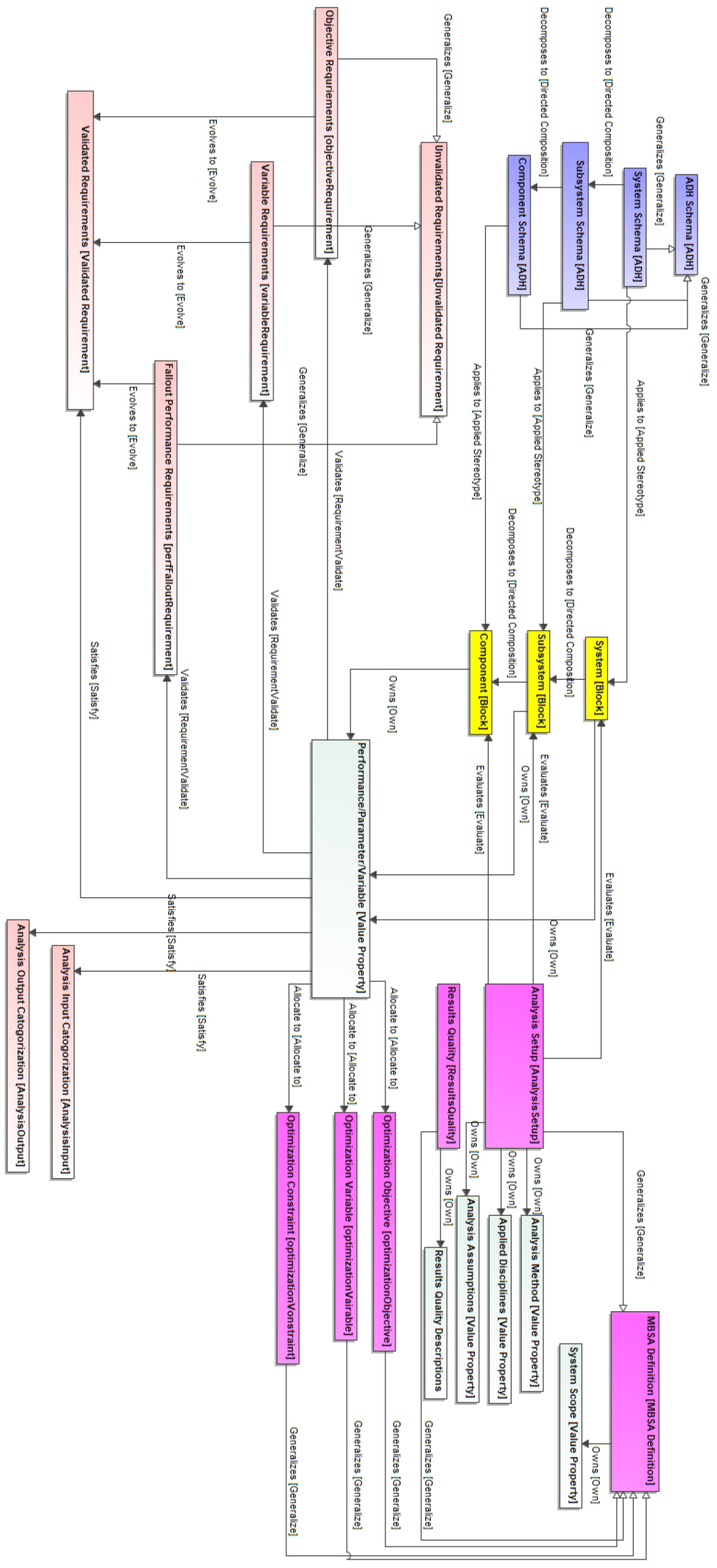

ADH Schema [ADH]
System Schema [ADH]
Subsystem Schema [ADH]
Component Schema [ADH]
Generalizes [Generalize]
Decomposes to [Directed Composition]
Applies to [Applied Stereotype]
System [Block]
Subsystem [Block]
Component [Block]
Owns [Own]
Evaluates [Evaluate]
Unvalidated Requirements[Unvalidated Requirement]
Objective Requriements [objectiveRequirement]
Variable Requirements [variableRequirement]
Fallout Performance Requirements [perfFalloutRequirement]
Validated Requirements [Validated Requirement]
Evolves to [Evolve]
Validates [RequirementValidate]
Satisfies [Satisfy]
Performance/Parameter/Variable [Value Property]
Analysis Output Catogorization [AnalysisOutput]
Analysis Input Catogorization [AnalysisInput]
Allocate to [Allocate to]
Optimization Constraint [optimizationVonstraint]
Optimization Variable [optimizationVariable]
Optimization Objective [optimizationObjective]
Results Quality [ResultsQuality]
Analysis Setup [AnalysisSetup]
Results Quality Descriptions
Analysis Assumptions [Value Property]
Applied Disciplines [Value Property]
Analysis Method [Value Property]
System Scope [Value Property]
MBSA Definition [MBSA Definition]

Figure 2 MBSE-MBSA Metamodel

Systems Architecture

Logical Architecture

It should be noted that the logical architecture of the airplane in this effort follows the sysML modeling convention, where <<block>> is used to represent the logical elements and the associated value properties are used to represent its performance/behavioral characteristics as well as parameters and variables. The corresponding metamodel is shown below.

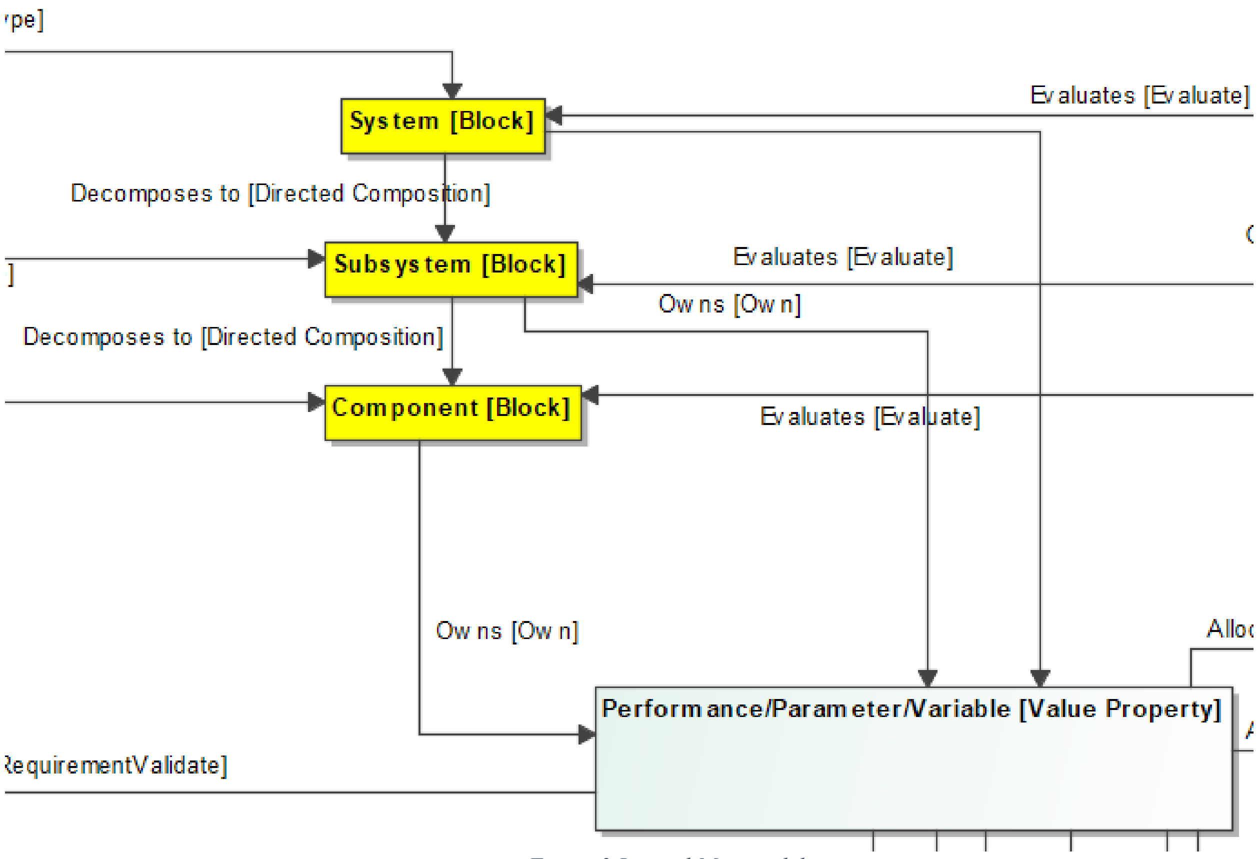


Figure 3 Logical Metamodel

Requirement Architecture

Besides the logical elements, another important aspect of this descriptive systems engineering model is the requirement, which is shown in the following figure.

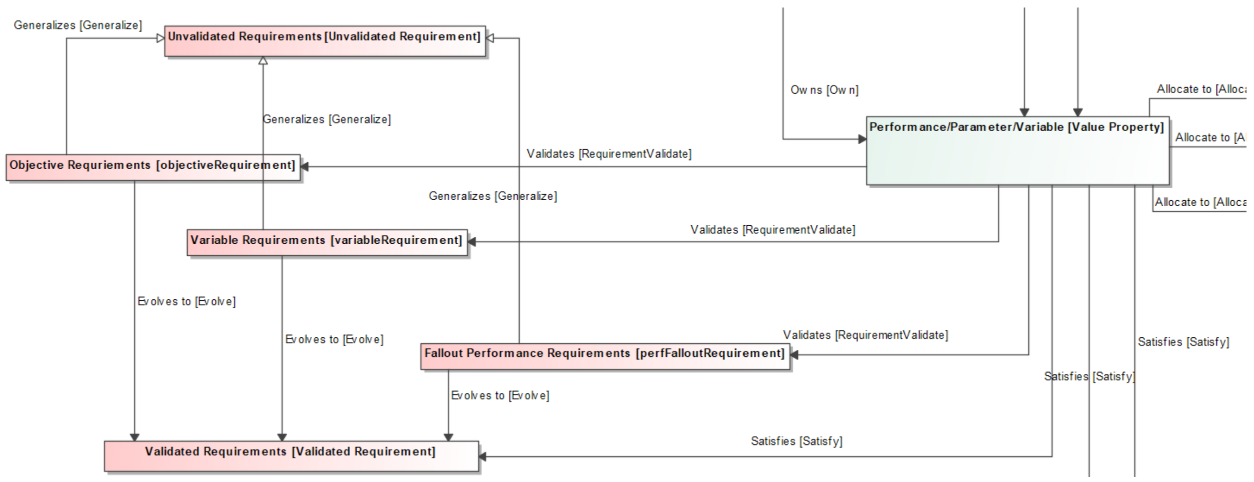

Generalizes [Generalize]
Unvalidated Requirements [Unvalidated Requirement]
Generalizes [Generalize]
Objective Requriements [objectiveRequirement]
Validates [RequirementValidate]
Owns [Own]
Performance/Parameter/Variable [Value Property]
Allocate to [Alloca
Allocate to [A
Allocate to [Alloca
Generalizes [Generalize]
Variable Requirements [variableRequirement]
Validates [RequirementValidate]
Evolves to [Evolve]
Evolves to [Evolve]
Validates [RequirementValidate]
Fallout Performance Requirements [perfFalloutRequirement]
Satisfies [Satisfy]
Satisfies [Satisfy]
Evolves to [Evolve]
Satisfies [Satisfy]
Validated Requirements [Validated Requirement]

Figure 4 Requirement Metamodel

It should be noted that the requirement metamodel is different from the basic usage of sysML due to the nature of the MDAO analysis cycle at airplane level. At the beginning of the analysis, the designers are supposed to know the required payload, range, and the regulation parts the airplane should comply to. However, the designers at this moment do have enough knowledge about the airplane performance. For example, the Maximum Takeoff Weight (MTOW) or fuel consumption for a specific mission are unknown. Therefore, it is impractical to directly set the corresponding requirements on MTOW or fuel capacity. Instead, an objective can be set to minimize MTOW or fuel capacity to optimize the airplane performance and explore the design space. However, such objectives are not real requirements, or they are defined as “Unvalidated Requirement” because there is not any analysis can validate them yet. Besides objective requirements, some of the requirements at lower system level are also unknown. For example, the wingspan or the maximum cruise thrust. The wingspan is a design requirement for the wing design, but the designer does not have such knowledge at the beginning of the design cycle, while the wingspan serves as a design variable at the airplane-level analysis. Therefore, such type of requirements are also not real requirements but another type of “Unvalidated Requirement”: Variable Requirements. For requirements such as the maximum cruise thrust, they are not design variables and they are obtained from the airplane performance analysis as fallout parameters. For such type of requirements, they are defined as “Fallout Performance Requirements” as another type of “Unvalidated Requirements”.
As the analysis cycle moves to the next system level, for example, at the wing design phase, the wingspan needs to become a real design requirement. In this case, the analysis from the analysis of previous cycle provides the data stored value properties and validate this unvalidated requirement, and then this requirement evolves to a validated requirement which means a real requirement such as “The wingspan shall be less than xxx ft”. In this scenario, special relations are needed to indicate the requirement validation process and the requirement evolution process, which are both illustrated in Figure 4. It should be noted here that “Unvalidated Requirement” and “Validated Requirement” are not necessarily implemented if there is no need for requirement evolution. In this case, the base requirement type should be used.

Besides the validation of requirements, the requirements also need to be verified since one of the objectives of the modeling effort is to show the requirements verification through simulation. Such verification relation is fulfilled by the “Satisfy” relation from sysML, which is a conventional sysML relation.

Aircraft Data Hierarchy

To be consistent with the disciplinary analysis tool, the Aircraft Data Hierarchy (ADH) is implemented in constructing the model structure. The ADH follows the Weight Breakdown Structure (WBS) of a few aerospace standards, which shows the decomposition of the systems from the top-level airplane down to subsystems and components. More details about ADH can be found in CLIN 001’s report. The metamodel regarding ADH can be found in the following figure.

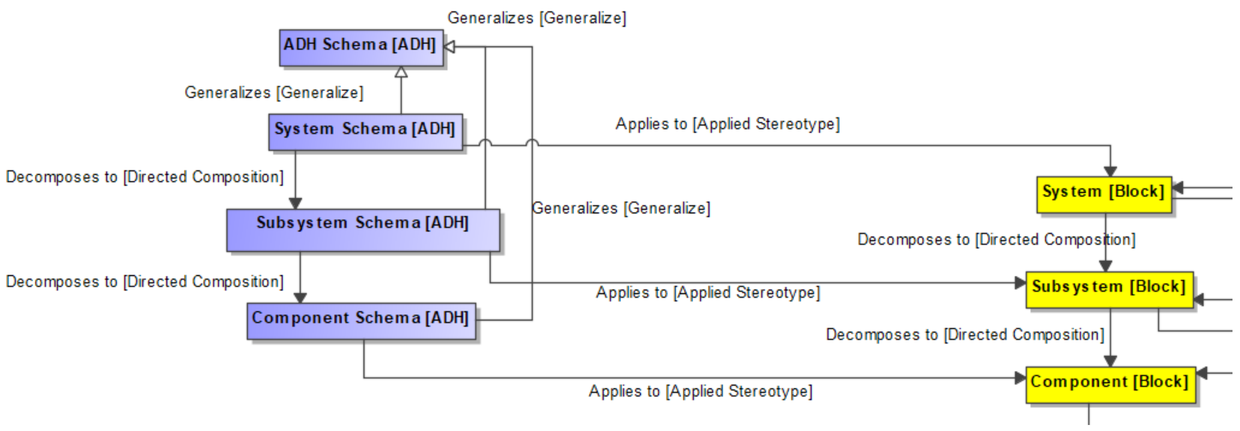

Generalizes [Generalize]
ADH Schema [ADH]
Generalizes [Generalize]
System Schema [ADH]
Applies to [Applied Stereotype]
Decomposes to [Directed Composition]
Subsystem Schema [ADH]
Generalizes [Generalize]
System [Block]
Decomposes to [Directed Composition]
Decomposes to [Directed Composition]
Applies to [Applied Stereotype]
Subsystem [Block]
Component Schema [ADH]
Decomposes to [Directed Composition]
Component [Block]
Applies to [Applied Stereotype]

Figure 5 ADH Metamodel

In the metamodel, a new base element of ADH schema is created, and then different ADH schema at different system levels are created by generalizing the base ADH schema. The implementation of the ADH schema is realized by applying the ADH schema as a stereotype to the logical blocks.

Analysis Traceability

Another important usage of the model is to trace the analysis and the design decision. The traceability metamodel is illustrated in Figure 6.

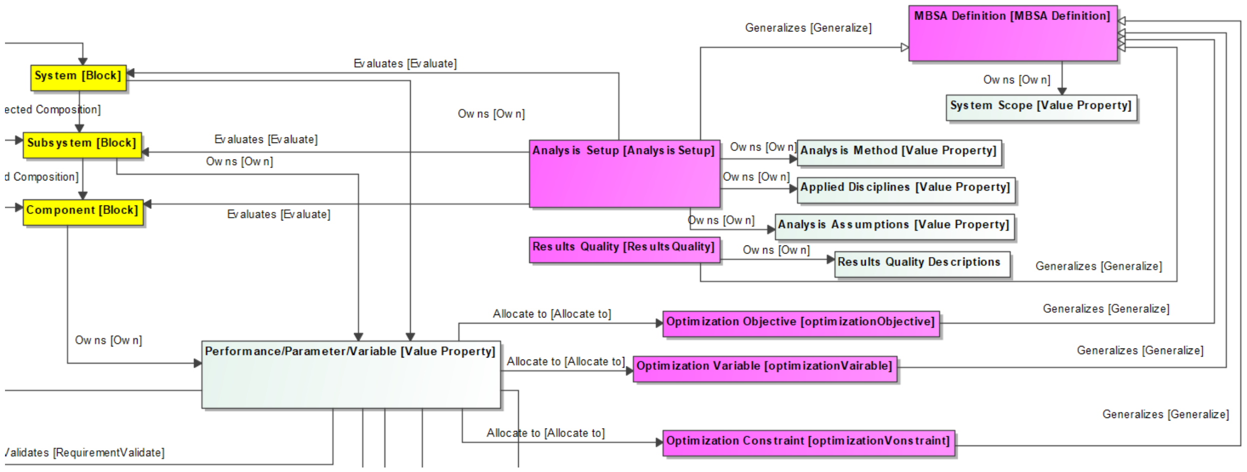

MBSA Definition [MBSA Definition]
Generalizes [Generalize]
System [Block]
Evaluates [Evaluate]
Owns [Own]
System Scope [Value Property]
ected Composition]
Owns [Own]
Subsystem [Block]
Evaluates [Evaluate]
Owns [Own]
Analysis Setup [Analysis Setup]
Owns [Own]
Analysis Method [Value Property]
d Composition]
Owns [Own]
Applied Disciplines [Value Property]
Component [Block]
Evaluates [Evaluate]
Owns [Own]
Analysis Assumptions [Value Property]
Results Quality [Results Quality]
Owns [Own]
Results Quality Descriptions
Generalizes [Generalize]
Allocate to [Allocate to]
Optimization Objective [optimizationObjective]
Generalizes [Generalize]
Owns [Own]
Performance/Parameter/Variable [Value Property]
Allocate to [Allocate to]
Optimization Variable [optimizationVairable]
Generalizes [Generalize]
Generalizes [Generalize]
Allocate to [Allocate to]
Optimization Constraint [optimizationVonstraint]
/alidates [RequirementValidate]

Figure 6 Traceability Metamodel

To realize the traceability between the analysis and the design, a base element as MBSA Definition is created, in which the scope of the system is also defined (e.g., airplane, wing, engine, etc.). Generalizing this base element, the analysis setup, result quality, optimization objective, optimization variable, and optimization constraint are also defined. The analysis setup refers to the setup of the disciplinary analysis, such as the algorithm, disciplinary assumptions, analysis version control, etc. The results quality refers to the convergence, solver tolerance, etc. The design decisions are assumed to be recorded in the system logical block in the corresponding value properties, and the traceability is established by creating the "evaluate" relation between the analysis setup and the logical blocks. It should be noted that the relation "evaluate" is also a default relation in sysML.

Model Sharing

The metamodel of the model sharing is illustrated in the following figure. This metamodel is very simple because the usage of the model sharing is to communicate with the partner about what the inputs the airplane analysis can provide to the ACM and what outputs the airplane analysis needs from the component model. In this sense, the only needed definition here is to categorize the values associated with each interface. The

Analysis Input Categorization and Analysis Output Categorization are customized type of the base type Requirement. Any value properties satisfying them are categorized as input and output of the collaborator's model accordingly.
It should be noted that the Internal Block Diagrams (IBDs) are also used to illustrate the interface between the airplane, ECS, and ACM. However, such IBDs follow the conventional sysML and NASA profile, so there is no need to add the IBD related part to the MBSE-MBSA metamodel.

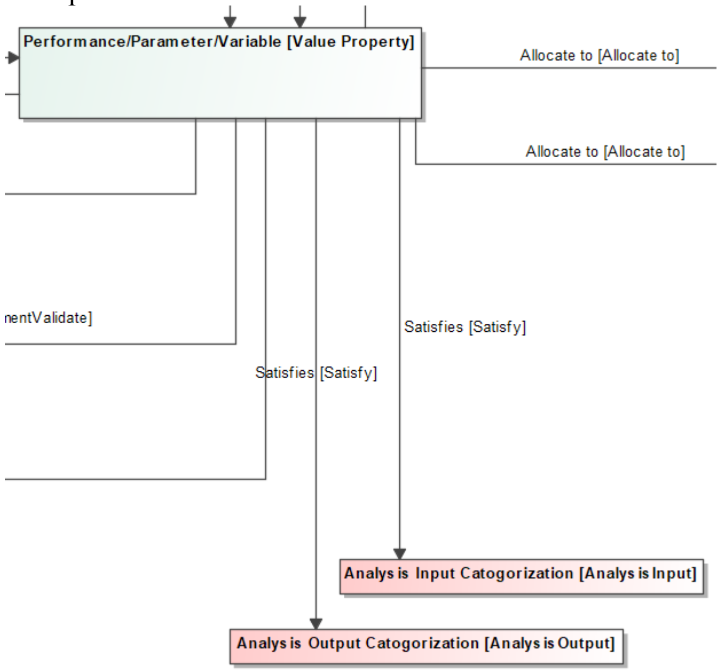

Performance/Parameter/Variable [Value Property]
Allocate to [Allocate to]
Allocate to [Allocate to]
nentValidate]
Satisfies [Satisfy]
Satisfies [Satisfy]
Analysis Input Catogorization [Analysis Input]
Analysis Output Catogorization [Analysis Output]

Figure 7 Model Sharing Metamodel

SysML Profile Development

The development of the profile is informed by the metamodel creation. The profile here refers to the stereotypes that are applied in the systems engineering model to realize our created metamodel. The profile and stereotypes developed in this program is in addition to the stereotypes developed by NASA [2].

ADH Profile

The ADH profile is illustrated in Figure 8.

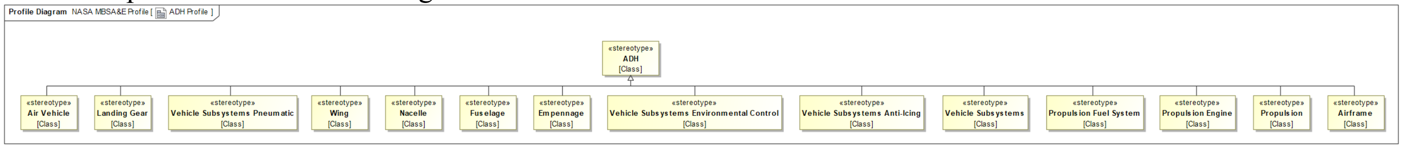
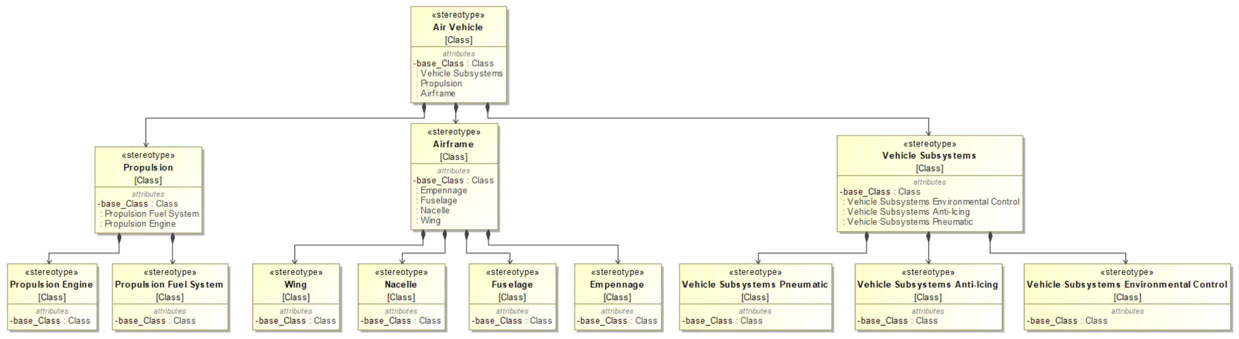

Profile Diagram NASA MBSA&E Profile [ADH Profile]
«stereotype» ADH [Class]
«stereotype» Air Vehicle [Class]
«stereotype» Landing Gear [Class]
«stereotype» Vehicle Subsystems Pneumatic [Class]
«stereotype» Wing [Class]
«stereotype» Nacelle [Class]
«stereotype» Fuselage [Class]
«stereotype» Empennage [Class]
«stereotype» Vehicle Subsystems Environmental Control [Class]
«stereotype» Vehicle Subsystems Anti-Icing [Class]
«stereotype» Vehicle Subsystems [Class]
«stereotype» Propulsion Fuel System [Class]
«stereotype» Propulsion Engine [Class]
«stereotype» Propulsion [Class]
«stereotype» Airframe [Class]

Figure 8 ADH Profile

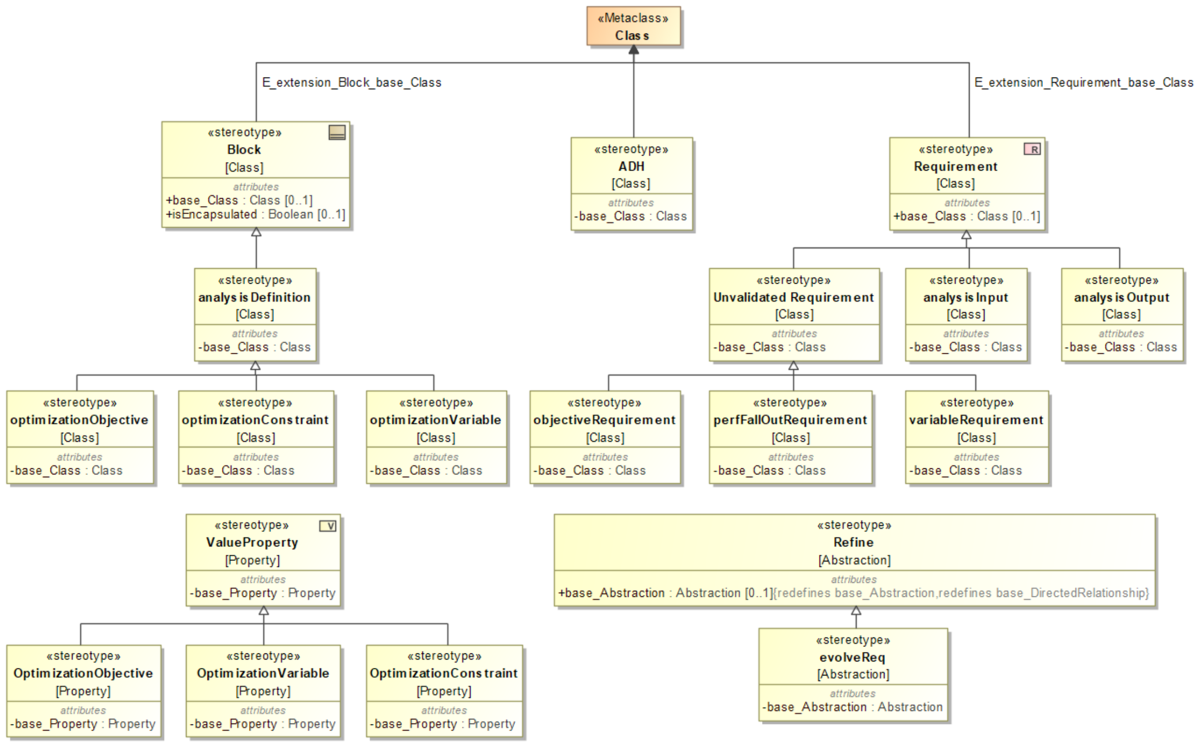
In this profile the top ADH stereotype is the base ADH schema stereotype while the other system-level, subsystem-level, and component-level ADH stereotypes are all generalizations of this base ADH schema stereotype. All the generalized ADH stereotypes follow the hierarchy defined in ADH as completed in CLIN 001. It should be noted here that not all systems from the ADH WBS have associated ADH stereotypes. Only the systems are used in the systems engineering model have associated ADH stereotypes. The ADH stereotypes themselves also follow the hierarchy as defined in CLIN 001, as illustrated by the following figure.


«stereotype» Air Vehicle [Class]
attributes
-base_Class : Class
: Vehicle Subsystems
: Propulsion
: Airframe
«stereotype» Propulsion [Class]
-base_Class : Class
: Propulsion Fuel System
: Propulsion Engine
«stereotype» Airframe [Class]
-base_Class : Class
: Empennage
: Fuselage
: Nacelle
: Wing
«stereotype» Vehicle Subsystems [Class]
-base_Class : Class
: Vehicle Subsystems Environmental Control
: Vehicle Subsystems Anti-Icing
: Vehicle Subsystems Pneumatic
«stereotype» Propulsion Engine [Class]
«stereotype» Propulsion Fuel System [Class]
«stereotype» Wing [Class]
«stereotype» Nacelle [Class]
«stereotype» Fuselage [Class]
«stereotype» Empennage [Class]
«stereotype» Vehicle Subsystems Pneumatic [Class]
«stereotype» Vehicle Subsystems Anti-Icing [Class]
«stereotype» Vehicle Subsystems Environmental Control [Class]
-base_Class : Class
Figure 9 ADH Schema Hierarchy

The methodology to implement the ADH stereotypes is discussed in more details in the next section.

MBSA&E Profile

The MBSA&E profile is shown below.


«Metaclass» Class
E_extension_Block_base_Class
E_extension_Requirement_base_Class
«stereotype» Block [Class]
+base_Class : Class [0..1]
+isEncapsulated : Boolean [0..1]
«stereotype» ADH [Class]
-base_Class : Class
«stereotype» Requirement [Class]
+base_Class : Class [0..1]
«stereotype» analysisDefinition [Class]
«stereotype» Unvalidated Requirement [Class]
«stereotype» analysisInput [Class]
«stereotype» analysisOutput [Class]
«stereotype» optimizationObjective [Class]
«stereotype» optimizationConstraint [Class]
«stereotype» optimizationVariable [Class]
«stereotype» objectiveRequirement [Class]
«stereotype» perfFallOutRequirement [Class]
«stereotype» variableRequirement [Class]
-base_Class : Class
«stereotype» ValueProperty [Property]
-base_Property : Property
«stereotype» Refine [Abstraction]
+base_Abstraction : Abstraction [0..1]{redefines base_Abstraction,redefines base_DirectedRelationship}
«stereotype» OptimizationObjective [Property]
«stereotype» OptimizationVariable [Property]
«stereotype» OptimizationConstraint [Property]
«stereotype» evolveReq [Abstraction]
-base_Abstraction : Abstraction
attributes
Figure 10 MBSA&E Profile

It can be seen that the ADH base stereotype is an extension of the Class which is a sysML Metaclass. The Analysis Definition which represents the Analysis Setup in the metamodel is a generalization of the basic block stereotype. This stereotype further is generalized to the variables, objectives, and constraints for the optimization. However, it should be noted that these three variables, objectives and constraints are still of the block stereotype, so they are not the actual value but the definition of these elements. Instead, the objectives, variables and constraints that are generalizing the base value property stereotype represent the actual values associated with these three quantities. One the right-hand side of this figure, the stereotypes for unvalidated requirement and the analysis input/out categorization are defined. The unvalidated requirement stereotype is further generalized to the objective requirement, performance fallout requirement, and variable requirement stereotypes. At the bottom the stereotype evolvReq represents the requirement evolution relation. It should be noted here again, as consistent with the development of the metamodel, if there is no need for such requirement evolution, then the stereotypes “Unvalidated Requirement” or “Validated Requirement” or the “evolveReq” relations are not necessarily needed.

Airplane Model Development

In this section the methodologies of the descriptive systems engineering model development are introduced. These methodologies included the aspects on the logical architecture, interface architecture (for model sharing), requirement architecture, along with the implementation of the ADH profile and the MBSA&E profile. At the end, the views created to show different systems engineering aspects of the models are also discussed.

Logical Architecture

The logical architecture refers to the composition of the system which is the airplane in this context. In the demonstration of the MBSE-MBSA digital connectivity, engine analysis was conducted as an example. Therefore, the major logical elements modeled in this project are around the propulsion system, which include the main engine, fuel system, and one of the major interfacing systems – environmental control system (ECS). The ECS systems engineering model is also used to demonstrate the model sharing practice with industrial partners (or suppliers) as an airplane integrator. Besides the propulsion logical models, the ECS models include the ECS itself, its subsystem (air distribution System and air cycle machine), its supporting system air management system, and its consumer wing anti-icing system (cabin is modeled together with the ECS). Besides the ECS and propulsion related aspects, all the other systems that are likely to be captured by the analysis framework Aviary[2] should also be modeled, which include propulsion (fuel system and engine), airframe (wing, fuselage, empennage, nacelle), landing gear system, and vehicle subsystem (ECS, air management system, wing anti-icing). The logical architecture is presented in the following figures.

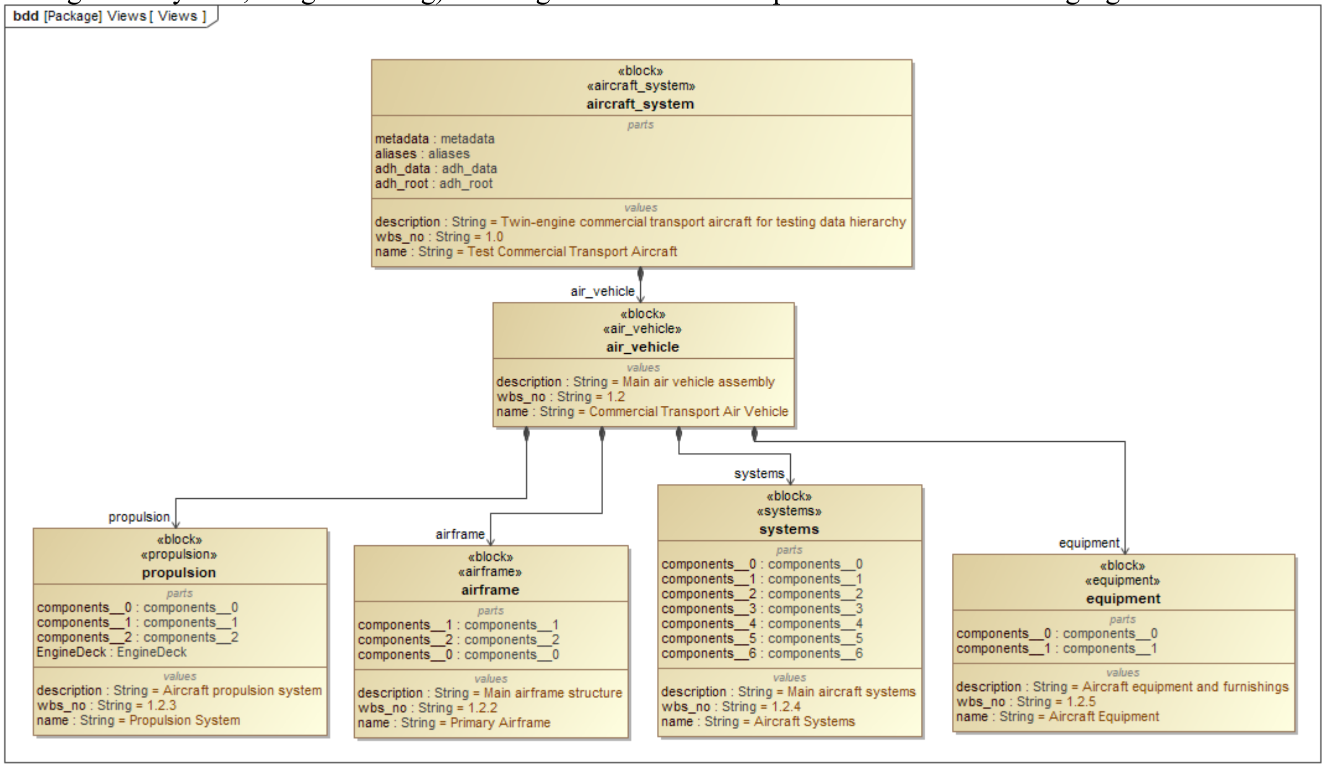
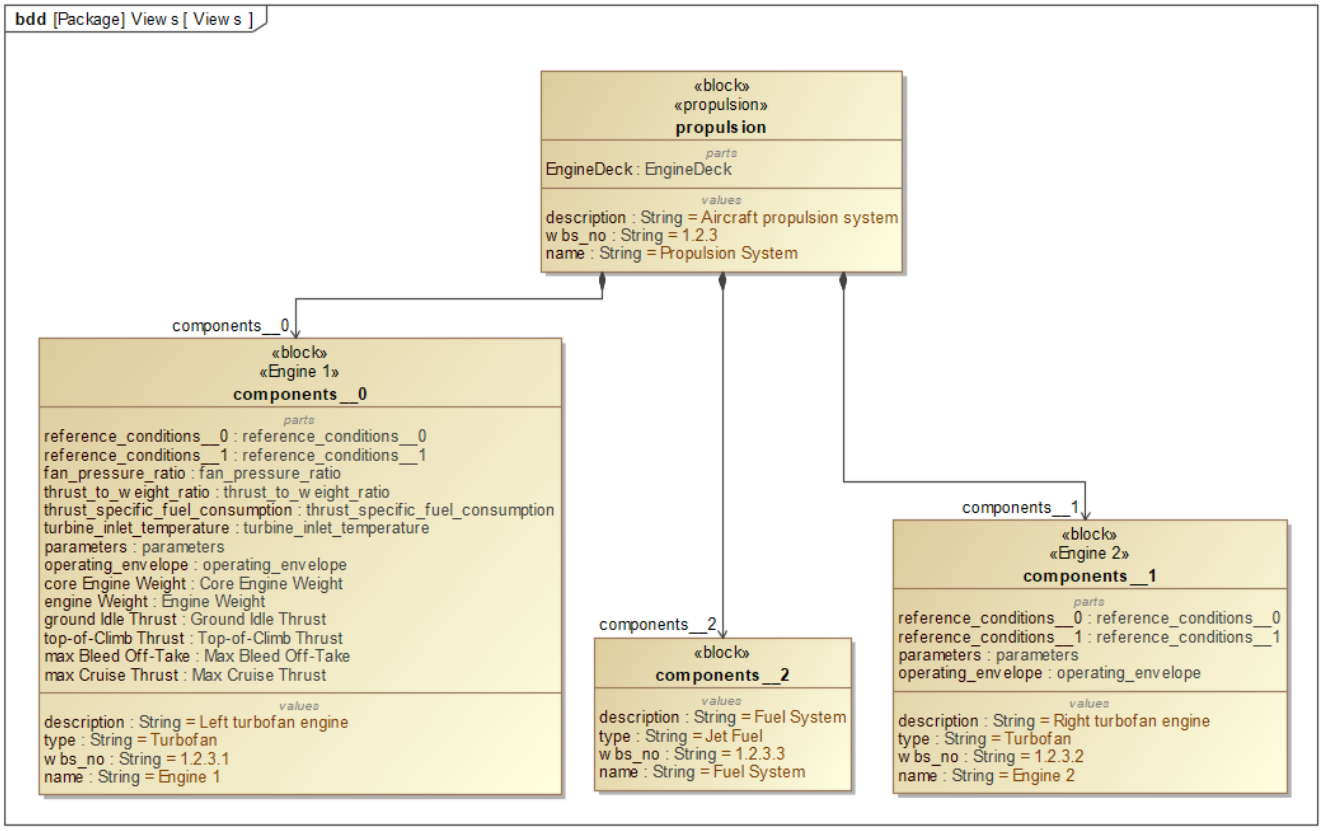

bdd [Package] Views [Views]
«block»
«aircraft_system»
aircraft_system
parts
metadata : metadata
aliases : aliases
adh_data : adh_data
adh_root : adh_root
values
description : String = Twin-engine commercial transport aircraft for testing data hierarchy
wbs_no : String = 1.0
name : String = Test Commercial Transport Aircraft
air_vehicle
«block»
«air_vehicle»
air_vehicle
values
description : String = Main air vehicle assembly
wbs_no : String = 1.2
name : String = Commercial Transport Air Vehicle
propulsion
«block»
«propulsion»
propulsion
parts
components__0 : components__0
components__1 : components__1
components__2 : components__2
EngineDeck : EngineDeck
values
description : String = Aircraft propulsion system
wbs_no : String = 1.2.3
name : String = Propulsion System
airframe
«block»
«airframe»
airframe
parts
components__1 : components__1
components__2 : components__2
components__0 : components__0
values
description : String = Main airframe structure
wbs_no : String = 1.2.2
name : String = Primary Airframe
systems
«block»
«systems»
systems
parts
components__0 : components__0
components__1 : components__1
components__2 : components__2
components__3 : components__3
components__4 : components__4
components__5 : components__5
components__6 : components__6
values
description : String = Main aircraft systems
wbs_no : String = 1.2.4
name : String = Aircraft Systems
equipment
«block»
«equipment»
equipment
parts
components__0 : components__0
components__1 : components__1
values
description : String = Aircraft equipment and furnishings
wbs_no : String = 1.2.5
name : String = Aircraft Equipment

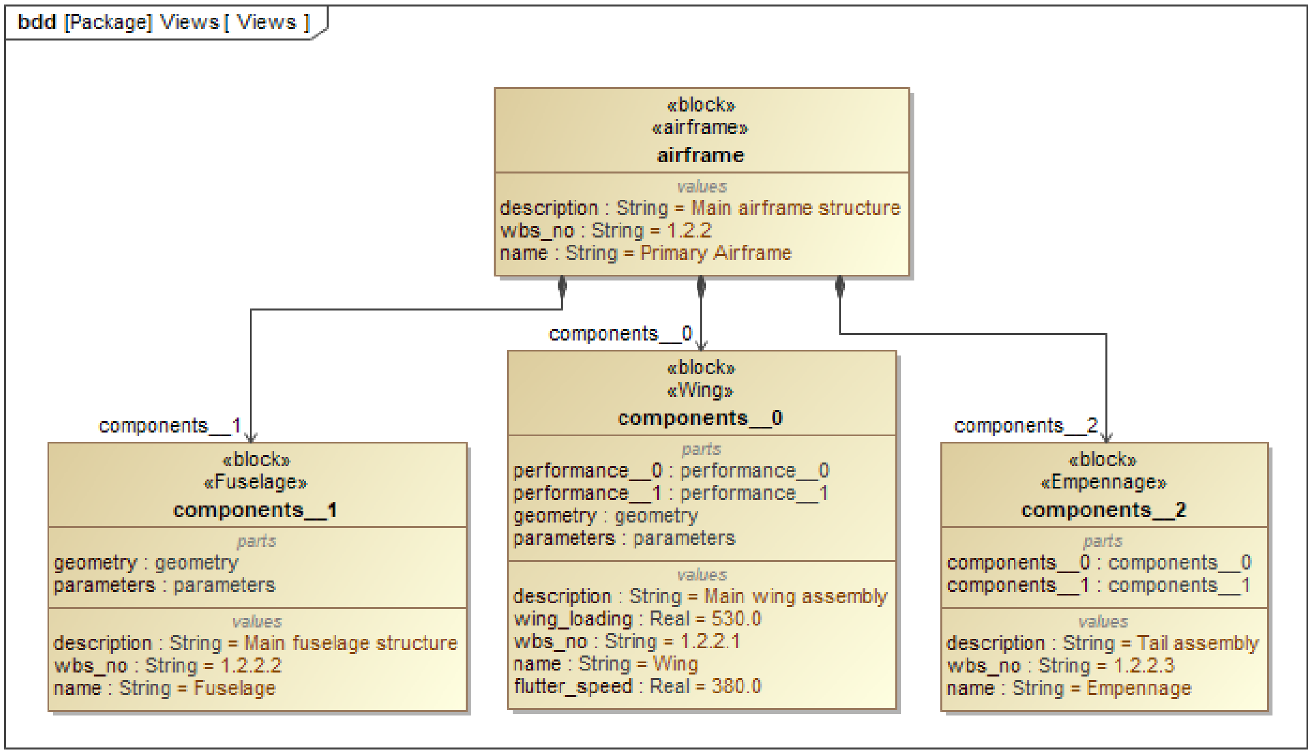
Figure 11 Logical Architecture – Airplane Level

[2] Aviary -- NASA's aircraft design tool, https://github.com/OpenMDAO/Aviary?tab=readme-ov-file



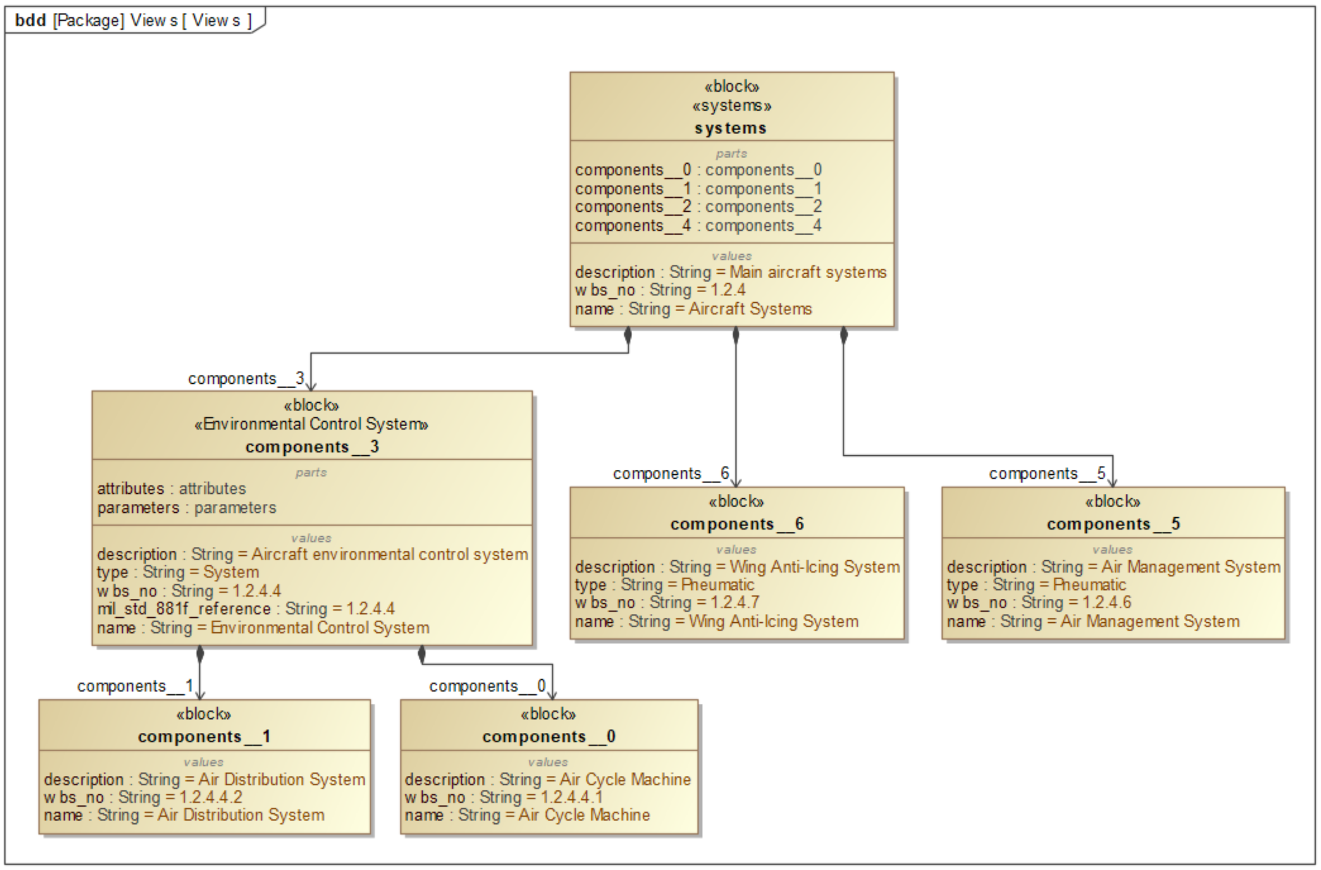
bdd [Package] Views [Views]
«block»
«propulsion»
propulsion
parts
EngineDeck : EngineDeck
values
description : String = Aircraft propulsion system
wbs_no : String = 1.2.3
name : String = Propulsion System
components__0
«block»
«Engine 1»
components__0
parts
reference_conditions__0 : reference_conditions__0
reference_conditions__1 : reference_conditions__1
fan_pressure_ratio : fan_pressure_ratio
thrust_to_weight_ratio : thrust_to_weight_ratio
thrust_specific_fuel_consumption : thrust_specific_fuel_consumption
turbine_inlet_temperature : turbine_inlet_temperature
parameters : parameters
operating_envelope : operating_envelope
core Engine Weight : Core Engine Weight
engine Weight : Engine Weight
ground Idle Thrust : Ground Idle Thrust
top-of-Climb Thrust : Top-of-Climb Thrust
max Bleed Off-Take : Max Bleed Off-Take
max Cruise Thrust : Max Cruise Thrust
values
description : String = Left turbofan engine
type : String = Turbofan
wbs_no : String = 1.2.3.1
name : String = Engine 1
components__2
«block»
components__2
values
description : String = Fuel System
type : String = Jet Fuel
wbs_no : String = 1.2.3.3
name : String = Fuel System
components__1
«block»
«Engine 2»
components__1
parts
reference_conditions__0 : reference_conditions__0
reference_conditions__1 : reference_conditions__1
parameters : parameters
operating_envelope : operating_envelope
values
description : String = Right turbofan engine
type : String = Turbofan
wbs_no : String = 1.2.3.2
name : String = Engine 2

Figure 12 Logical Architecture – Propulsion



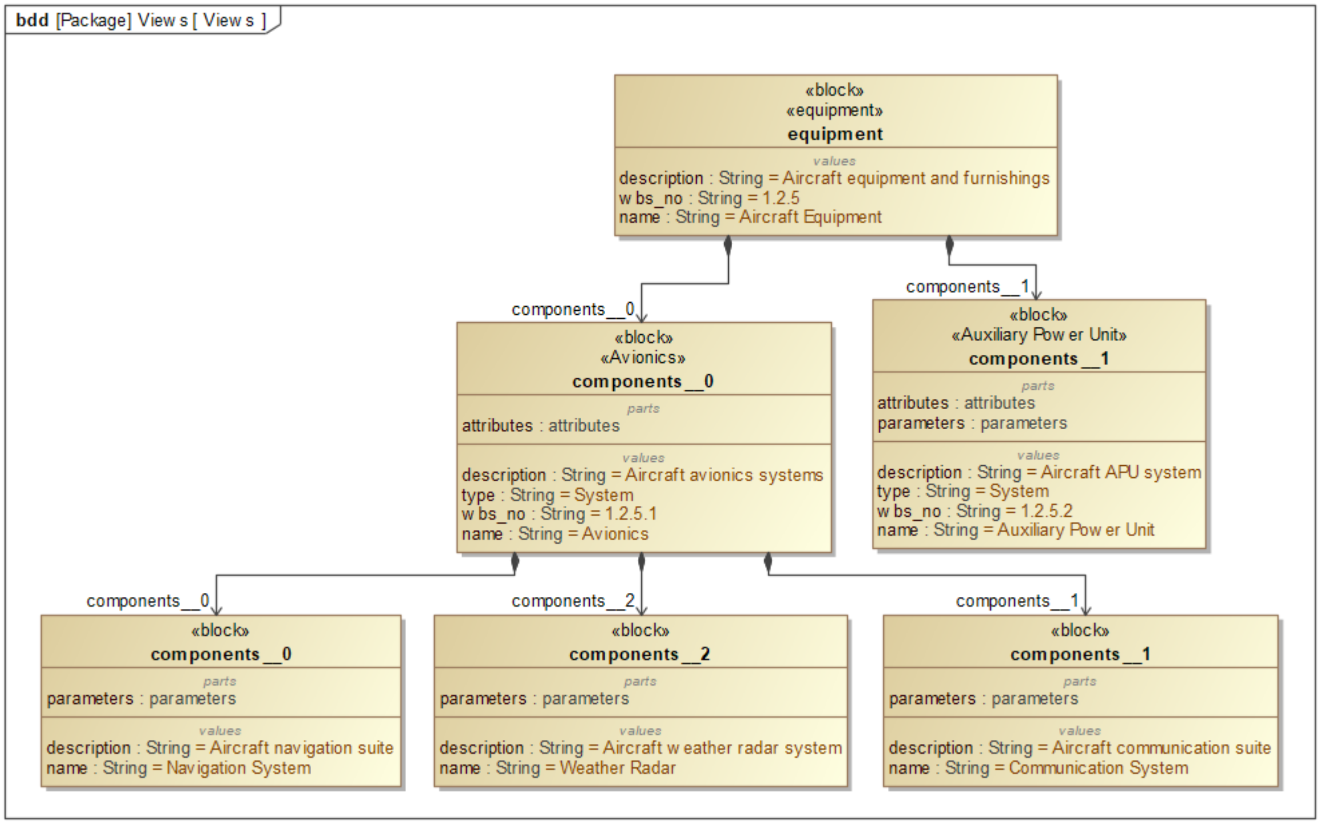
bdd [Package] Views [Views]
«block»
«airframe»
airframe
values
description : String = Main airframe structure
wbs_no : String = 1.2.2
name : String = Primary Airframe
components__0
«block»
«Wing»
components__0
parts
performance__0 : performance__0
performance__1 : performance__1
geometry : geometry
parameters : parameters
values
description : String = Main wing assembly
wing_loading : Real = 530.0
wbs_no : String = 1.2.2.1
name : String = Wing
flutter_speed : Real = 380.0
components__1
«block»
«Fuselage»
components__1
parts
geometry : geometry
parameters : parameters
values
description : String = Main fuselage structure
wbs_no : String = 1.2.2.2
name : String = Fuselage
components__2
«block»
«Empennage»
components__2
parts
components__0 : components__0
components__1 : components__1
values
description : String = Tail assembly
wbs_no : String = 1.2.2.3
name : String = Empennage

Figure 13 Logical Architecture – Airframe



bdd [Package] Views [Views]
«block» «systems» systems
parts
components__0 : components__0
components__1 : components__1
components__2 : components__2
components__4 : components__4
values
description : String = Main aircraft systems
wbs_no : String = 1.2.4
name : String = Aircraft Systems
components__3
«block» «Environmental Control System» components__3
parts
attributes : attributes
parameters : parameters
values
description : String = Aircraft environmental control system
type : String = System
wbs_no : String = 1.2.4.4
mil_std_881f_reference : String = 1.2.4.4
name : String = Environmental Control System
components__6
«block» components__6
values
description : String = Wing Anti-Icing System
type : String = Pneumatic
wbs_no : String = 1.2.4.7
name : String = Wing Anti-Icing System
components__5
«block» components__5
values
description : String = Air Management System
type : String = Pneumatic
wbs_no : String = 1.2.4.6
name : String = Air Management System
components__1
«block» components__1
values
description : String = Air Distribution System
wbs_no : String = 1.2.4.4.2
name : String = Air Distribution System
components__0
«block» components__0
values
description : String = Air Cycle Machine
wbs_no : String = 1.2.4.4.1
name : String = Air Cycle Machine

Figure 14 Logical Architecture - Environmental Control



bdd [Package] Views [Views]
«block» «equipment» equipment
values
description : String = Aircraft equipment and furnishings
wbs_no : String = 1.2.5
name : String = Aircraft Equipment
components__0
«block» «Avionics» components__0
parts
attributes : attributes
values
description : String = Aircraft avionics systems
type : String = System
wbs_no : String = 1.2.5.1
name : String = Avionics
components__1
«block» «Auxiliary Power Unit» components__1
parts
attributes : attributes
parameters : parameters
values
description : String = Aircraft APU system
type : String = System
wbs_no : String = 1.2.5.2
name : String = Auxiliary Power Unit
components__0
«block» components__0
parts
parameters : parameters
values
description : String = Aircraft navigation suite
name : String = Navigation System
components__2
«block» components__2
parts
parameters : parameters
values
description : String = Aircraft weather radar system
name : String = Weather Radar
components__1
«block» components__1
parts
parameters : parameters
values
description : String = Aircraft communication suite
name : String = Communication System

Figure 15 Logical Architecture - Equipment

In terms of the organization of the ADH structure, the value properties that represent the performance, behavioral, or design characteristics are not directly owned by the system logical component. Instead, they are owned by Parameter and Performance blocks, which are shown in the following figures. Such deviations from the traditional sysML modeling method is because systems engineering models in this contract is supposed to be consistent with the four branches established by the ADH structure.

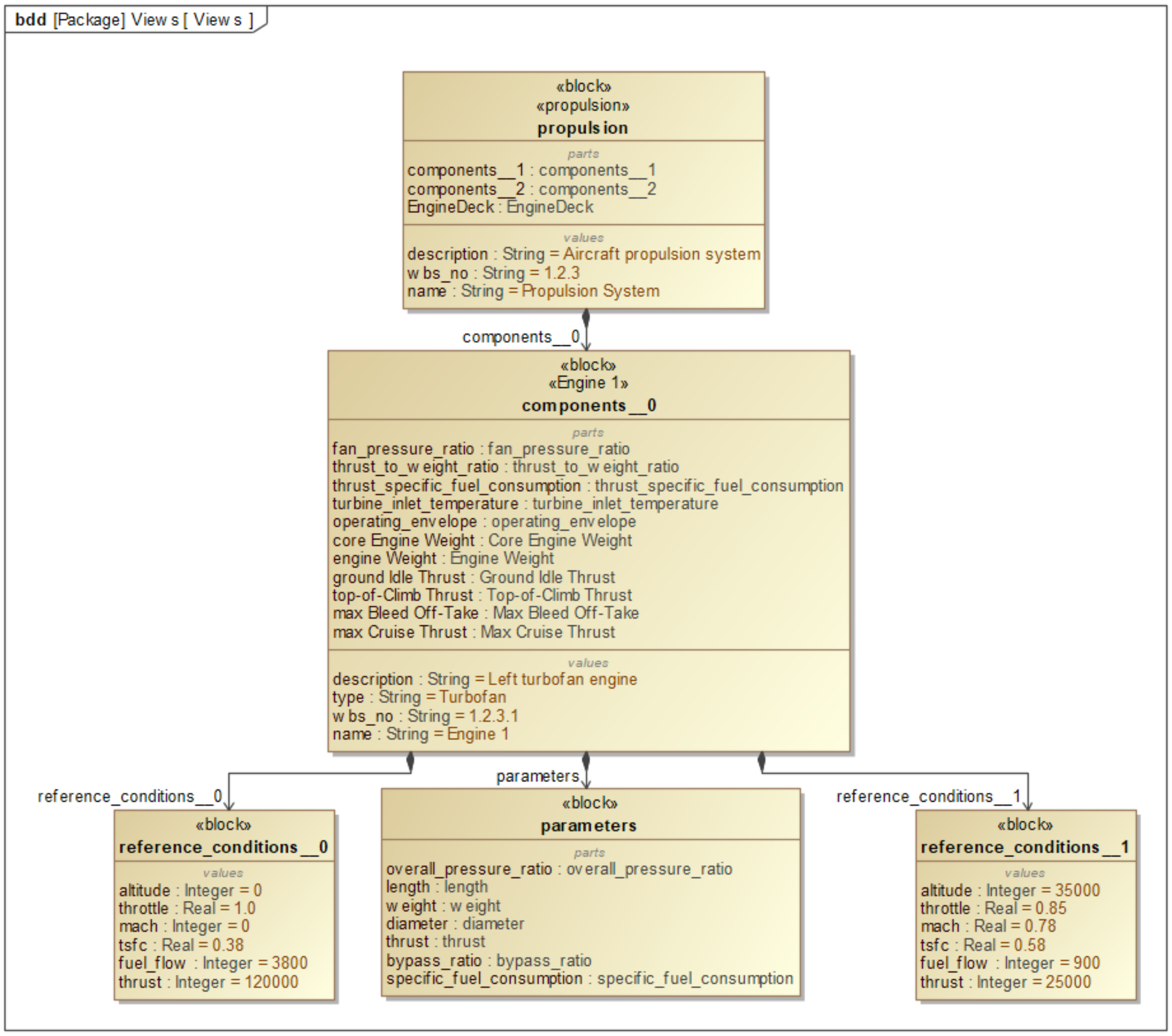

bdd [Package] Views [Views]
«block»
«propulsion»
propulsion
parts
components__1 : components__1
components__2 : components__2
EngineDeck : EngineDeck
values
description : String = Aircraft propulsion system
wbs_no : String = 1.2.3
name : String = Propulsion System
components__0
«block»
«Engine 1»
components__0
parts
fan_pressure_ratio : fan_pressure_ratio
thrust_to_weight_ratio : thrust_to_weight_ratio
thrust_specific_fuel_consumption : thrust_specific_fuel_consumption
turbine_inlet_temperature : turbine_inlet_temperature
operating_envelope : operating_envelope
core Engine Weight : Core Engine Weight
engine Weight : Engine Weight
ground Idle Thrust : Ground Idle Thrust
top-of-Climb Thrust : Top-of-Climb Thrust
max Bleed Off-Take : Max Bleed Off-Take
max Cruise Thrust : Max Cruise Thrust
values
description : String = Left turbofan engine
type : String = Turbofan
wbs_no : String = 1.2.3.1
name : String = Engine 1
reference_conditions__0
parameters
reference_conditions__1
«block»
reference_conditions__0
values
altitude : Integer = 0
throttle : Real = 1.0
mach : Integer = 0
tsfc : Real = 0.38
fuel_flow : Integer = 3800
thrust : Integer = 120000
«block»
parameters
parts
overall_pressure_ratio : overall_pressure_ratio
length : length
weight : weight
diameter : diameter
thrust : thrust
bypass_ratio : bypass_ratio
specific_fuel_consumption : specific_fuel_consumption
«block»
reference_conditions__1
values
altitude : Integer = 35000
throttle : Real = 0.85
mach : Real = 0.78
tsfc : Real = 0.58
fuel_flow : Integer = 900
thrust : Integer = 25000

Figure 16 Engine Related Value Properties

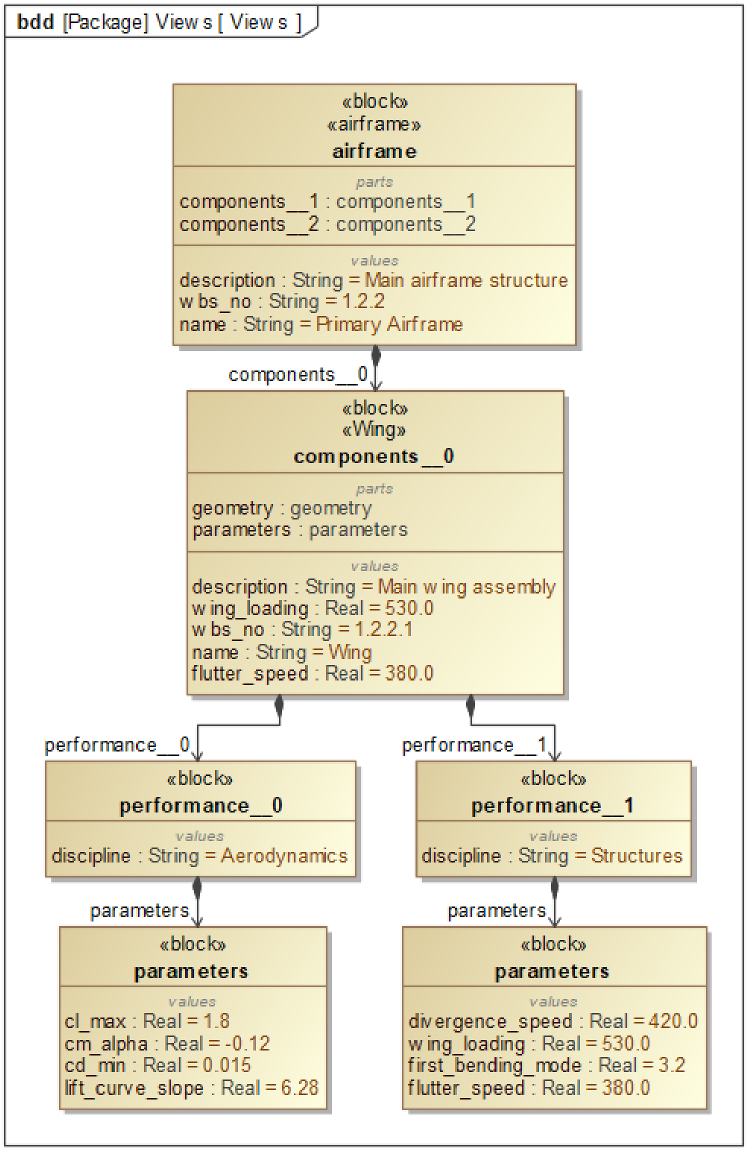

bdd [Package] Views [Views]
«block»
«airframe»
airframe
parts
components__1 : components__1
components__2 : components__2
values
description : String = Main airframe structure
wbs_no : String = 1.2.2
name : String = Primary Airframe
components__0
«block»
«Wing»
components__0
parts
geometry : geometry
parameters : parameters
values
description : String = Main wing assembly
wing_loading : Real = 530.0
wbs_no : String = 1.2.2.1
name : String = Wing
flutter_speed : Real = 380.0
performance__0
performance__1
«block»
performance__0
values
discipline : String = Aerodynamics
«block»
performance__1
values
discipline : String = Structures
parameters
parameters
«block»
parameters
values
cl_max : Real = 1.8
cm_alpha : Real = -0.12
cd_min : Real = 0.015
lift_curve_slope : Real = 6.28
«block»
parameters
values
divergence_speed : Real = 420.0
wing_loading : Real = 530.0
first_bending_mode : Real = 3.2
flutter_speed : Real = 380.0

Figure 17 Airframe Related Value Properties

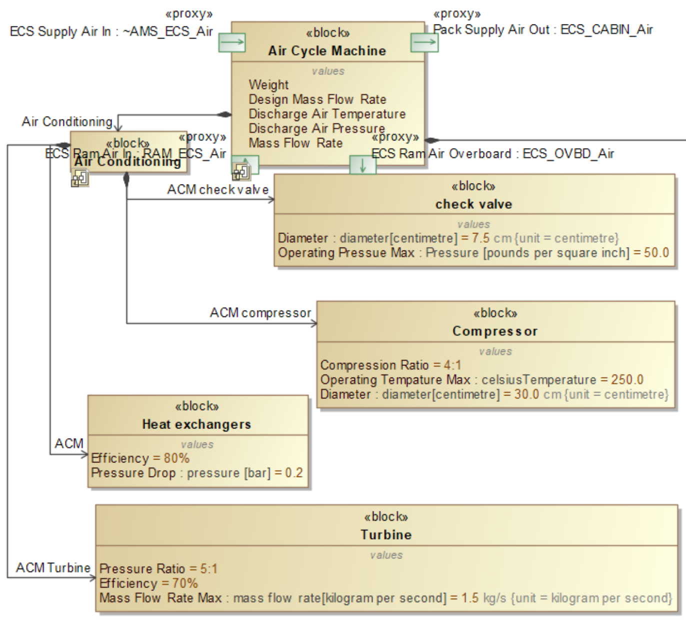

«proxy»
ECS Supply Air In : ~AMS_ECS_Air
«block»
Air Cycle Machine
values
Weight
Design Mass Flow Rate
Discharge Air Temperature
Discharge Air Pressure
Mass Flow Rate
«proxy»
Pack Supply Air Out : ECS_CABIN_Air
Air Conditioning
«block»
Air Conditioning
«proxy»
ECS Ram Air In : RAM_ECS_Air
«proxy»
ECS Ram Air Overboard : ECS_OVBD_Air
Fan System
«block»
Fan System
values
Material = Titanium Alloys
ACM check valve
«block»
check valve
values
Diameter : diameter[centimetre] = 7.5 cm {unit = centimetre}
Operating Pressue Max : Pressure [pounds per square inch] = 50.0
ACM fan
«block»
Fan
values
Spin Rate Max : RPM = 30000
Spin Rate Min : RPM = 0
Diameter : diameter[centimetre] = 30.0 cm {unit = centimetre}
ACM compressor
«block»
Compressor
values
Compression Ratio = 4:1
Operating Tempature Max : celsiusTemperature = 250.0
Diameter : diameter[centimetre] = 30.0 cm {unit = centimetre}
ACM shaft
«block»
3 wheel shaft
values
Length : length[centimetre] = 22.0 cm {unit = centimetre}
Diameter : diameter[centimetre] = 2.0 cm {unit = centimetre}
Torque Max : torque[new ton metre] = 50.0 N.m {unit = new ton metre}
Torque Min : torque[new ton metre] = 10.0 N.m {unit = new ton metre}
ACM
«block»
Heat exchangers
values
Efficiency = 80%
Pressure Drop : pressure [bar] = 0.2
ACM Turbine
«block»
Turbine
values
Pressure Ratio = 5:1
Efficiency = 70%
Mass Flow Rate Max : mass flow rate[kilogram per second] = 1.5 kg/s {unit = kilogram per second}

Figure 18 Logical Architecture - Supplier's Shared Air Cycle Machine Model

Requirement Architecture

The requirement architecture should start from the top-level business requirement. This business requirement shall define the class of the airplane to be developed (payload and range), and also the associated regulations (e.g., 14 CFR PART 25). The business requirement comes from market analysis, which is out of the scope, so only a placeholder is created in the current model.

From the top-level business requirement, the payload and range requirements are derived, together with two unvalidated requirements: minimize fuel consumption and minimize MTOW. These unvalidated requirements are of the stereotype “Objective Requirement” as created in the MBSA&E profile. Besides, a requirement on the mission definition can also be derived from the business requirement, and this mission definition can further derive to operational constraints such as cruise altitude, taxi max speed, minimum climb rate, etc.

The regulation is also part of the requirement. In this study, the regulation on ventilation, extended operations (ETOPS), and one-engine-inoperative are considered, because of the selected trade study use case. These regulations can further derive to the requirements that set the minimum air supply from the ECS and the minimum bleed supply from the engine. The requirements on the ECS are communicated with the industrial partner on their ACM development, and in the supplier’s ACM model, the component-level requirements are developed based on the ECS-level requirements. When the ACM model is incorporated into the main airplane model, the requirement tree becomes complete from the top-level business requirement, down to system level, and then to subsystem level, and then finally to the component level.

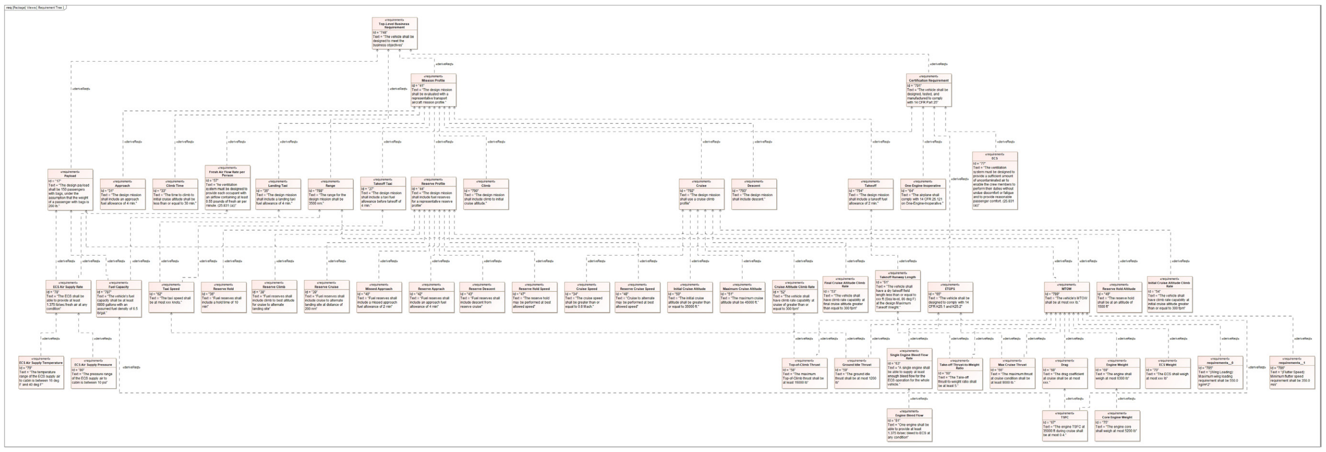
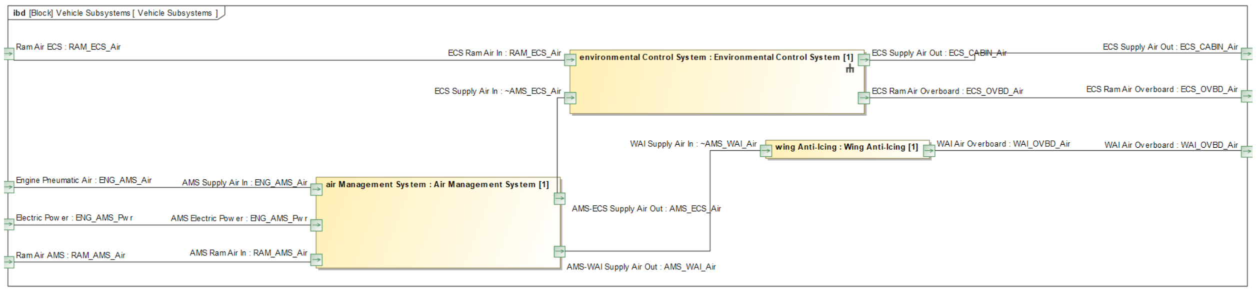


Figure 19 Requirement Tree

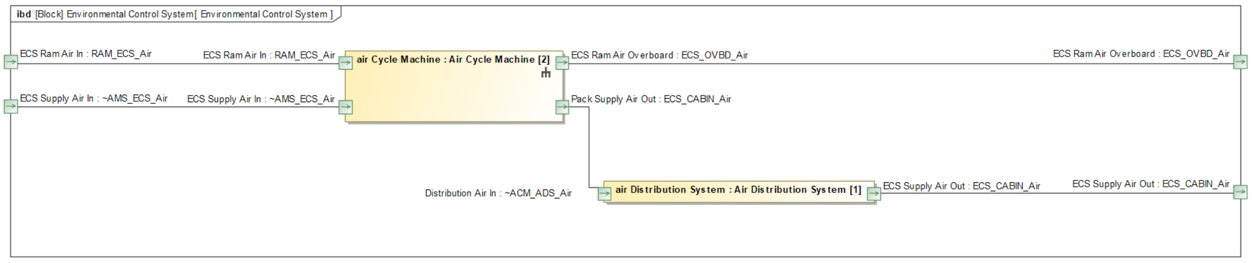
Interface Architecture

The interface is not used for any of the disciplinary analysis in MBSA but only used for the communications with industrial partner on the interface between subsystem-level logical elements and component-level logical elements. In this context, they are subsystems and components that are related to ECS operations. The corresponding interface architectures are presented below.



ibd [Block] Vehicle Subsystems [Vehicle Subsystems]
Ram Air ECS : RAM_ECS_Air
ECS Ram Air In : RAM_ECS_Air
environmental Control System : Environmental Control System [1]
ECS Supply Air Out : ECS_CABIN_Air
ECS Supply Air Out : ECS_CABIN_Air
ECS Supply Air In : ~AMS_ECS_Air
ECS Ram Air Overboard : ECS_OVBD_Air
ECS Ram Air Overboard : ECS_OVBD_Air
WAI Supply Air In : ~AMS_WAI_Air
wing Anti-Icing : Wing Anti-Icing [1]
WAI Air Overboard : WAI_OVBD_Air
WAI Air Overboard : WAI_OVBD_Air
Engine Pneumatic Air : ENG_AMS_Air
AMS Supply Air In : ENG_AMS_Air
air Management System : Air Management System [1]
AMS-ECS Supply Air Out : AMS_ECS_Air
Electric Power : ENG_AMS_Pwr
AMS Electric Power : ENG_AMS_Pwr
Ram Air AMS : RAM_AMS_Air
AMS Ram Air In : RAM_AMS_Air
AMS-WAI Supply Air Out : AMS_WAI_Air

Figure 20 Subsystem-level Collector Interface



ibd [Block] Environmental Control System [Environmental Control System]
ECS Ram Air In : RAM_ECS_Air
ECS Ram Air In : RAM_ECS_Air
air Cycle Machine : Air Cycle Machine [2]
ECS Ram Air Overboard : ECS_OVBD_Air
ECS Ram Air Overboard : ECS_OVBD_Air
ECS Supply Air In : ~AMS_ECS_Air
ECS Supply Air In : ~AMS_ECS_Air
Pack Supply Air Out : ECS_CABIN_Air
Distribution Air In : ~ACM_ADS_Air
air Distribution System : Air Distribution System [1]
ECS Supply Air Out : ECS_CABIN_Air
ECS Supply Air Out : ECS_CABIN_Air

Figure 21 ECS Interface

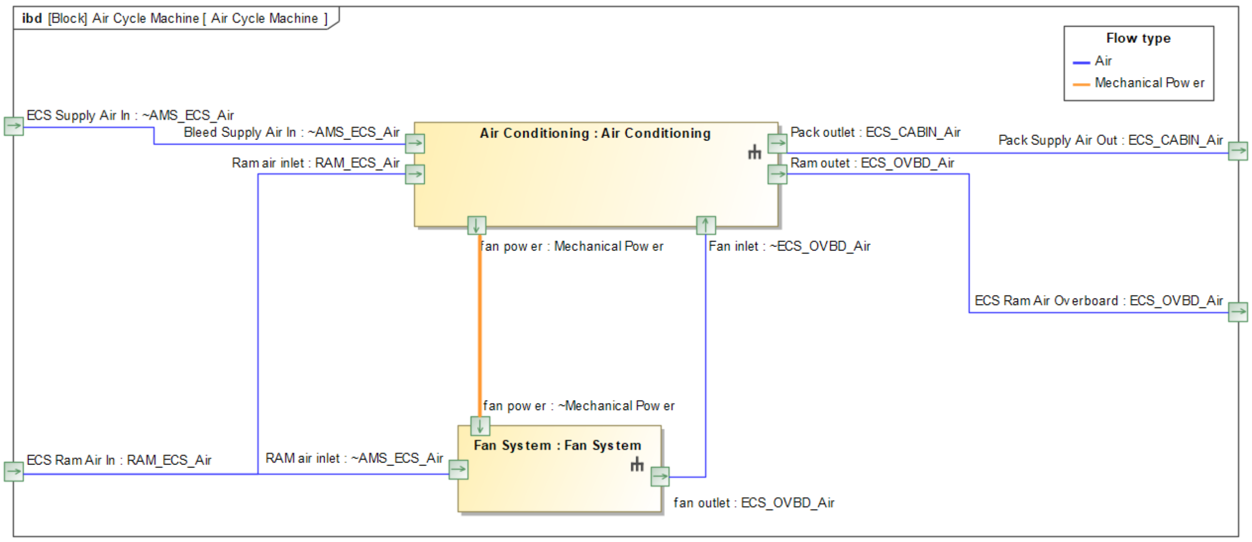

ibd [Block] Air Cycle Machine [Air Cycle Machine]
Flow type
Air
Mechanical Pow er
ECS Supply Air In : ~AMS_ECS_Air
Bleed Supply Air In : ~AMS_ECS_Air
Ram air inlet : RAM_ECS_Air
Air Conditioning : Air Conditioning
Pack outlet : ECS_CABIN_Air
Ram outet : ECS_OVBD_Air
Pack Supply Air Out : ECS_CABIN_Air
fan pow er : Mechanical Pow er
Fan inlet : ~ECS_OVBD_Air
ECS Ram Air Overboard : ECS_OVBD_Air
fan pow er : ~Mechanical Pow er
Fan System : Fan System
ECS Ram Air In : RAM_ECS_Air
RAM air inlet : ~AMS_ECS_Air
fan outlet : ECS_OVBD_Air

Figure 22 ACM Interface

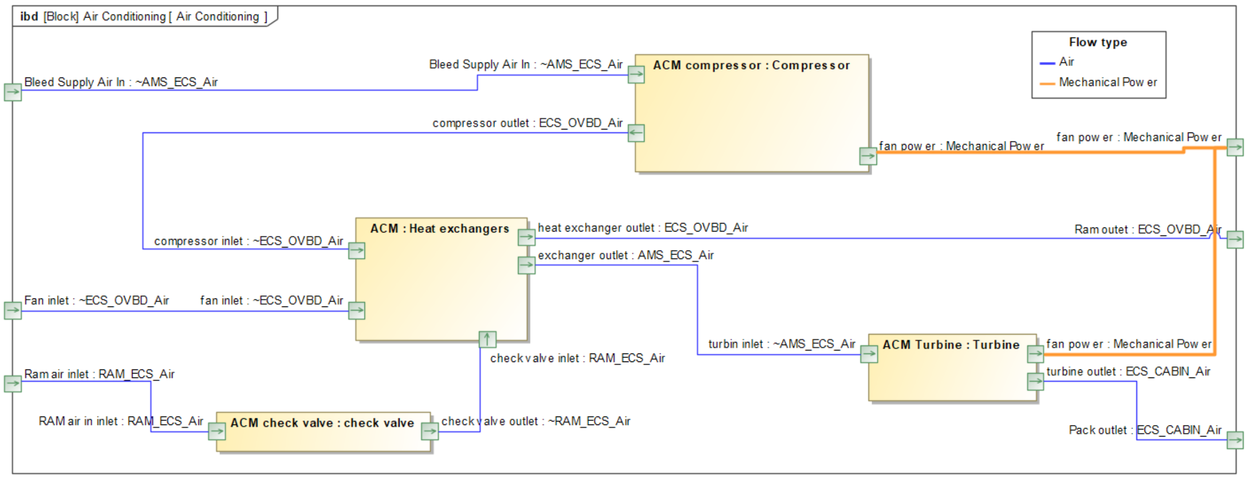

ibd [Block] Air Conditioning [Air Conditioning]
Flow type
Air
Mechanical Pow er
Bleed Supply Air In : ~AMS_ECS_Air
Bleed Supply Air In : ~AMS_ECS_Air
ACM compressor : Compressor
compressor outlet : ECS_OVBD_Air
fan pow er : Mechanical Pow er
fan pow er : Mechanical Pow er
compressor inlet : ~ECS_OVBD_Air
ACM : Heat exchangers
heat exchanger outlet : ECS_OVBD_Air
exchanger outlet : AMS_ECS_Air
Ram outet : ECS_OVBD_Air
Fan inlet : ~ECS_OVBD_Air
fan inlet : ~ECS_OVBD_Air
check valve inlet : RAM_ECS_Air
turbin inlet : ~AMS_ECS_Air
ACM Turbine : Turbine
fan pow er : Mechanical Pow er
turbine outlet : ECS_CABIN_Air
Ram air inlet : RAM_ECS_Air
RAM air in inlet : RAM_ECS_Air
ACM check valve : check valve
check valve outlet : ~RAM_ECS_Air
Pack outlet : ECS_CABIN_Air

Figure 23 ACM Component Interface (1)

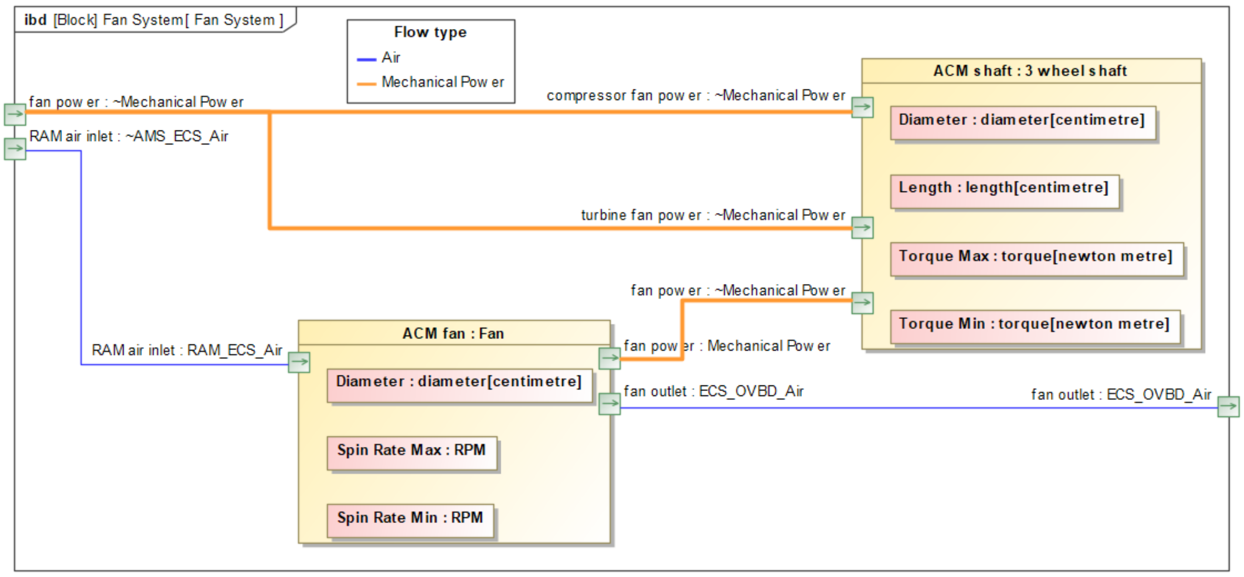

ibd [Block] Fan System[Fan System]
Flow type
Air
Mechanical Pow er
fan pow er : ~Mechanical Pow er
RAM air inlet : ~AMS_ECS_Air
compressor fan pow er : ~Mechanical Pow er
ACM shaft : 3 wheel shaft
Diameter : diameter[centimetre]
Length : length[centimetre]
Torque Max : torque[newton metre]
Torque Min : torque[newton metre]
turbine fan pow er : ~Mechanical Pow er
fan pow er : ~Mechanical Pow er
fan pow er : Mechanical Pow er
RAM air inlet : RAM_ECS_Air
ACM fan : Fan
Diameter : diameter[centimetre]
Spin Rate Max : RPM
Spin Rate Min : RPM
fan outlet : ECS_OVBD_Air
fan outlet : ECS_OVBD_Air

Figure 24 ACM Component Interface (2)

One important note here is that the interface architecture shall be developed from the highest-level system that is applicable to the use case. As the high-level interface architecture is matured, and then the information/material flow will be inherited by the lower-level interface architecture. If inconsistencies are found at low-level interface architectures, it usually indicates incorrect modeling on high-level interfaces. The usage of these interface architecture in support of model sharing communication will be discussed in the next section with more details.

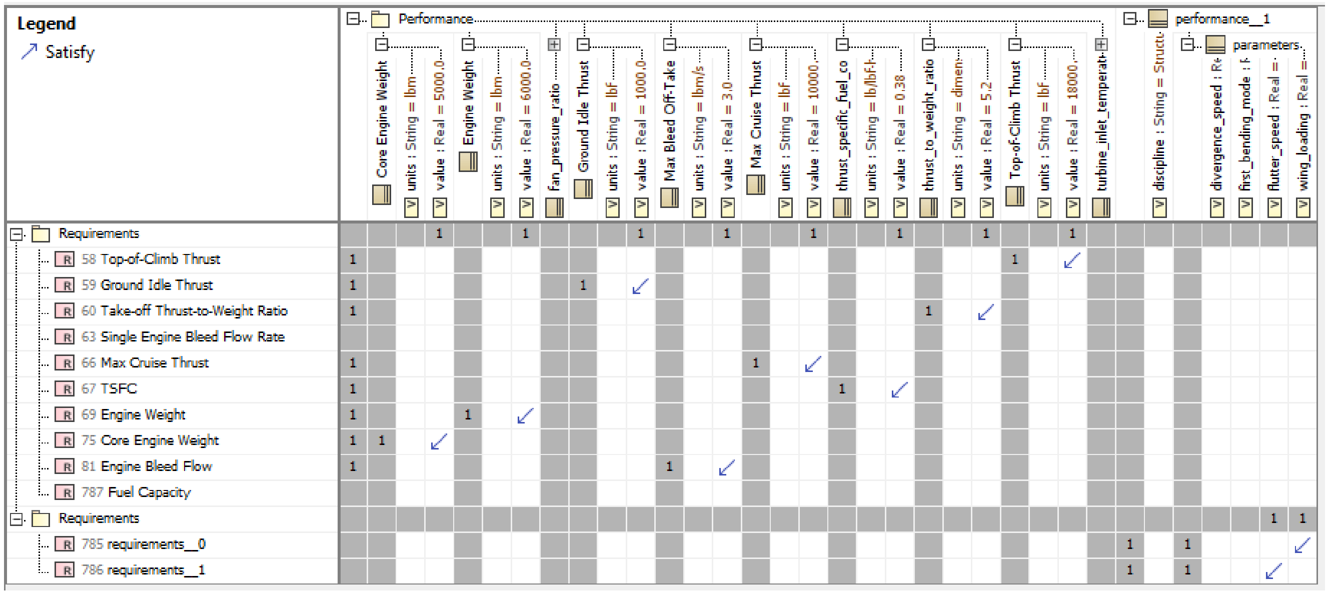
Please note that the interface architecture is only included in the model that is used to support model sharing but not in the model for MBSE-MBSA digital connectivity.

Views

Besides the models of the systems' logical, requirements, and performance aspects, systems engineering views are also needed to understand the relations among these systems aspects. The systems engineering views in this model include the following artifacts:

1. Airplane Architecture (Block Definition Diagram)
2. Requirement Dependency (Dependency Matrix)
3. Requirement Tree (Requirement Diagram)
4. Requirement Table (Requirement Table)
5. Requirement Satisfy (Satisfy Requirement Matrix)
6. Requirement Verification (Verification Requirement Matrix)

Airplane Architecture is a Block Definition Diagram (BDD) which is to show the decomposition relations among airplane, system, subsystem, and components. Besides the decomposition relations, the ownership of the performance and parameters are also represented. These BDDs have been shown in previous figures, as in Figure 11 - Figure 17. Requirement Dependency Matrix is to illustrate the parent-child relations among the requirements, which provides a convenient way to check the dependency among each requirement. The Requirement Tree is a requirement diagram to show the derivation paths of a certain requirement, as shown in Figure 19. The Requirement Table simply provides a tabular format of all the requirements associated with the systems engineering model. The Requirement Satisfy Matrix provides what value properties, or what parameter/performance matrices are evaluated quantitatively whether certain requirements are met or not. The Requirement Verification Matrix represents what logical components/system definitions are used to verify certain requirements. These view artifacts are shown in the figures below:



Legend
↗ Satisfy

	Performance	Core Engine Weight	units : String = lbm	value : Real = 5000.0	Engine Weight	units : String = lbm	value : Real = 6000.0	fan_pressure_ratio	Ground Idle Thrust	units : String = lbf	value : Real = 1000.0	Max Bleed Off-Take	units : String = lbm/s	value : Real = 3.0	Max Cruise Thrust	units : String = lbf	value : Real = 10000.	thrust_specific_fuel_co	units : String = lb/lbf-h	value : Real = 0.38	thrust_to_weight_ratio	units : String = dimen	value : Real = 5.2	Top-of-Climb Thrust	units : String = lbf	value : Real = 18000.	turbine_inlet_temperat	performance__1	discipline : String = Structu	parameters	divergence_speed : Re	first_bending_mode : F	flutter_speed : Real =	wing_loading : Real =
Requirements				1			1				1			1			1			1			1			1								
58 Top-of-Climb Thrust	1																							1		↙								
59 Ground Idle Thrust	1								1		↙																							
60 Take-off Thrust-to-Weight Ratio	1																				1		↙											
63 Single Engine Bleed Flow Rate																																		
66 Max Cruise Thrust	1														1		↙																	
67 TSFC	1																	1		↙														
69 Engine Weight	1				1		↙																											
75 Core Engine Weight	1	1		↙																														
81 Engine Bleed Flow	1											1		↙																				
787 Fuel Capacity																																		
Requirements																																	1	1
785 requirements__0																												1		1				↙
786 requirements__1																												1		1			↙	

Figure 25 Requirement Satisfy Matrix

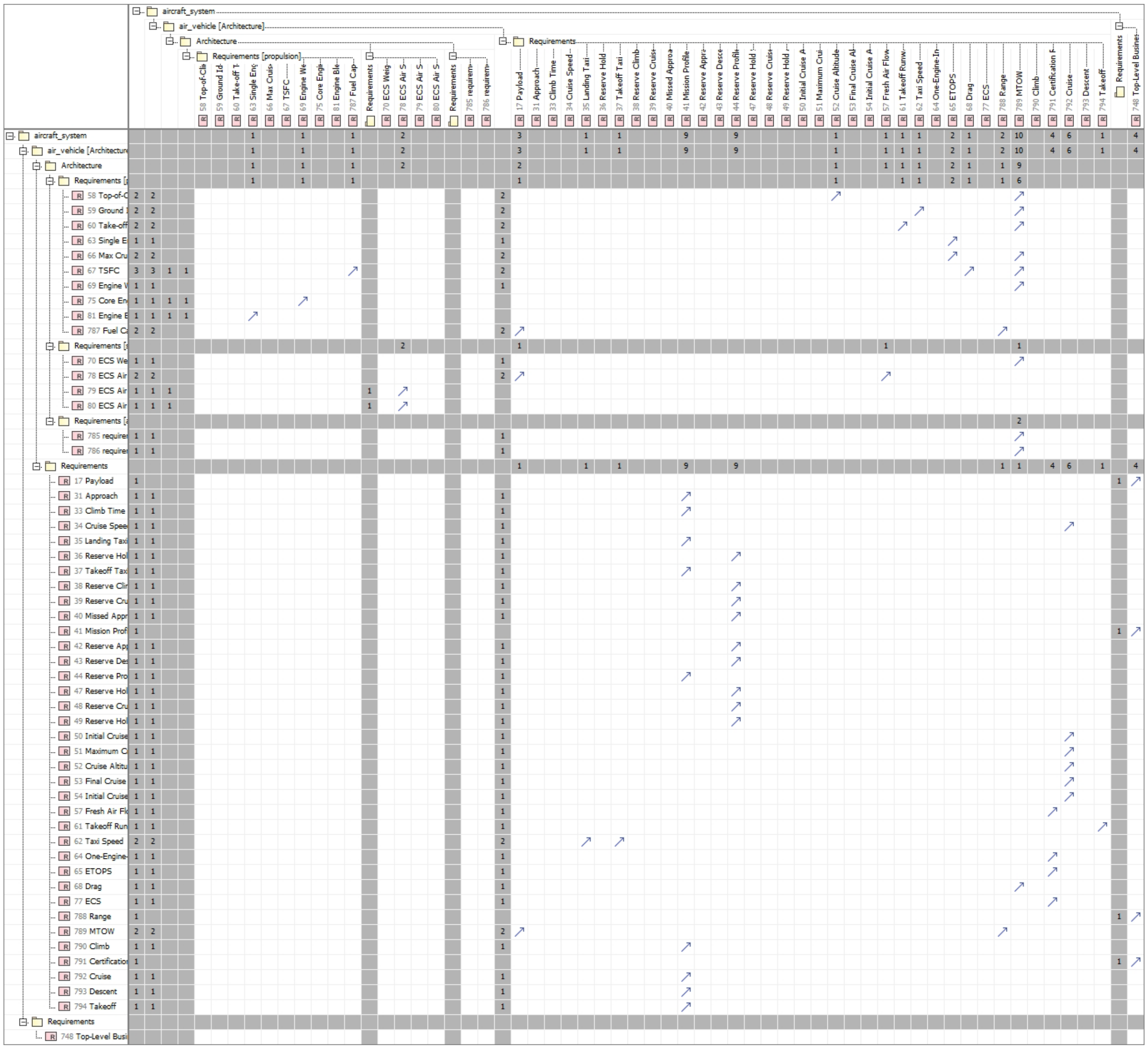


Figure 26 Requirement Dependency Matrix

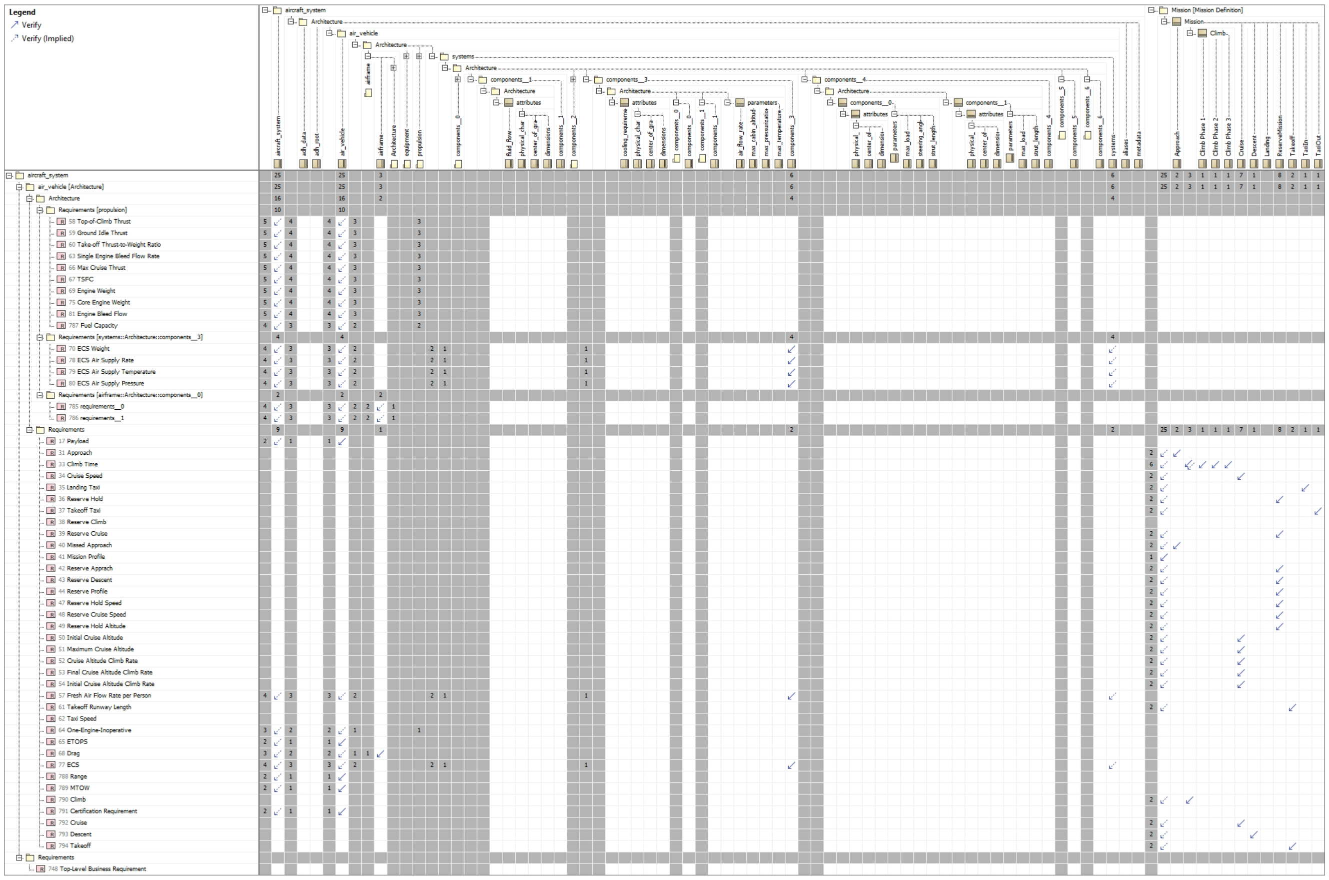

Legend
Verify
Verify (Implied)
aircraft_system
air_vehicle [Architecture]
Architecture
Requirements [propulsion]
58 Top-of-Climb Thrust
59 Ground Idle Thrust
60 Take-off Thrust-to-Weight Ratio
63 Single Engine Bleed Flow Rate
66 Max Cruise Thrust
67 TSFC
69 Engine Weight
75 Core Engine Weight
81 Engine Bleed Flow
787 Fuel Capacity
Requirements [systems::Architecture::components__3]
70 ECS Weight
78 ECS Air Supply Rate
79 ECS Air Supply Temperature
80 ECS Air Supply Pressure
Requirements [airframe::Architecture::components__0]
785 requirements__0
786 requirements__1
Requirements
17 Payload
31 Approach
33 Climb Time
34 Cruise Speed
35 Landing Taxi
36 Reserve Hold
37 Takeoff Taxi
38 Reserve Climb
39 Reserve Cruise
40 Missed Approach
41 Mission Profile
42 Reserve Apprach
43 Reserve Descent
44 Reserve Profile
47 Reserve Hold Speed
48 Reserve Cruise Speed
49 Reserve Hold Altitude
50 Initial Cruise Altitude
51 Maximum Cruise Altitude
52 Cruise Altitude Climb Rate
53 Final Cruise Altitude Climb Rate
54 Initial Cruise Altitude Climb Rate
57 Fresh Air Flow Rate per Person
61 Takeoff Runway Length
62 Taxi Speed
64 One-Engine-Inoperative
65 ETOPS
68 Drag
77 ECS
788 Range
789 MTOW
790 Climb
791 Certification Requirement
792 Cruise
793 Descent
794 Takeoff
Requirements
748 Top-Level Business Requirement
Mission [Mission Definition]
Mission
Climb
Approach
Climb Phase 1
Climb Phase 2
Climb Phase 3
Cruise
Descent
Landing
ReserveMission
Takeoff
TaxiIn
TaxiOut

Figure 27 Requirement Verification Matrix

Requirement table is purely a text-based table, and it is too big to include here. Please refer to the deliverable for the requirement table view.

SysML Model to Support Model Sharing

Scope of Model Sharing

There are two major objectives of the model sharing: 1. Communicate the information that is needed by the airplane to complete the analysis and the associated interfaces; 2. Share the requirements that need to be verified through simulation with collaborators to support their development of requirements at component level.
To fulfill these modeling objectives, interface architecture is created (previous section) to support the interface information communication, and requirements are developed at the ECS level.

Interface Information for Sharing

Initially the information needed by the airplane analysis module is collected from the analysis tools. Then this information is used to create the top-level interface architecture around ECS, and the information and material flows are developed to mature the interfaces at the ECS level, as shown in Figure 20. After the interfaces are matured at the ECS level, the internal interface architecture of ECS is then being developed as illustrated in Figure 21. The architecture represented by Figure 21 is the lowest-level interface architecture that the airplane integrator can develop. Therefore, Figure 21 needs to be communicated to the collaborator for further development.

When Figure 21 is received by Collins Aerospace, the interface architecture within the ACM is being developed. More details are added to the ACM as well as the components within the ACM, as shown in Figure 22, Figure 23, and Figure 24. These developed ACM models are then shared by to Boeing so that Boeing can integrate such information back to the airplane.

The definition of the I/O status of each interface is through the application of the "analysisInput" and "analysisOutput" stereotypes in MBSA&E profile.

System Requirement for Sharing

Besides the communication of interface information, the supplier also needs to develop their own component requirements. The starting point is the requirements at the ECS level, which are illustrated below as an example.

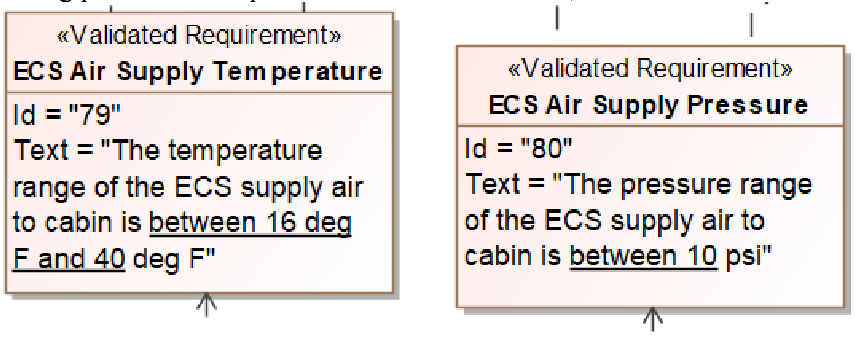

«Validated Requirement»
ECS Air Supply Temperature
Id = "79"
Text = "The temperature range of the ECS supply air to cabin is between 16 deg F and 40 deg F"
«Validated Requirement»
ECS Air Supply Pressure
Id = "80"
Text = "The pressure range of the ECS supply air to cabin is between 10 psi"

Figure 28 ECS Requirement Example

Then these two requirements further derived to the ACM-level requirement as shown below, which will be used for integration verification.

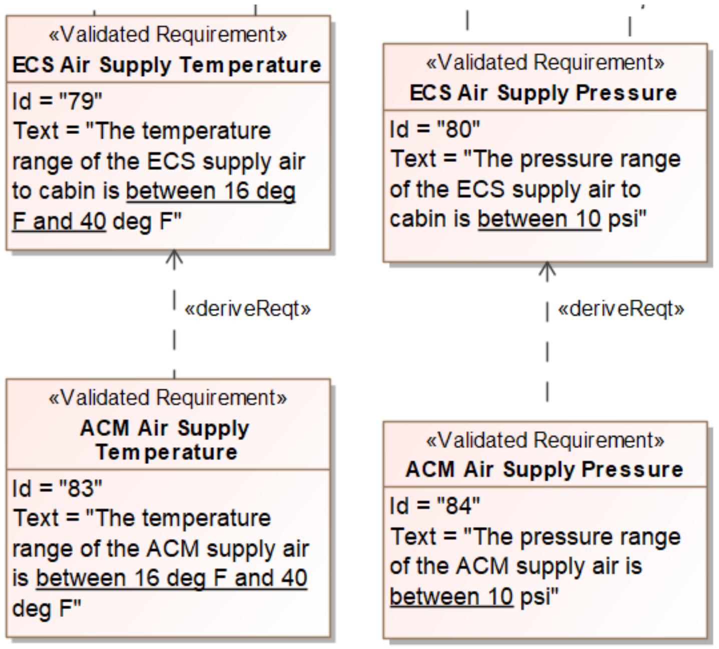

«Validated Requirement»
ECS Air Supply Temperature
Id = "79"
Text = "The temperature range of the ECS supply air to cabin is between 16 deg F and 40 deg F"
«Validated Requirement»
ECS Air Supply Pressure
Id = "80"
Text = "The pressure range of the ECS supply air to cabin is between 10 psi"
«deriveReqt»
«deriveReqt»
«Validated Requirement»
ACM Air Supply Temperature
Id = "83"
Text = "The temperature range of the ACM supply air is between 16 deg F and 40 deg F"
«Validated Requirement»
ACM Air Supply Pressure
Id = "84"
Text = "The pressure range of the ACM supply air is between 10 psi"

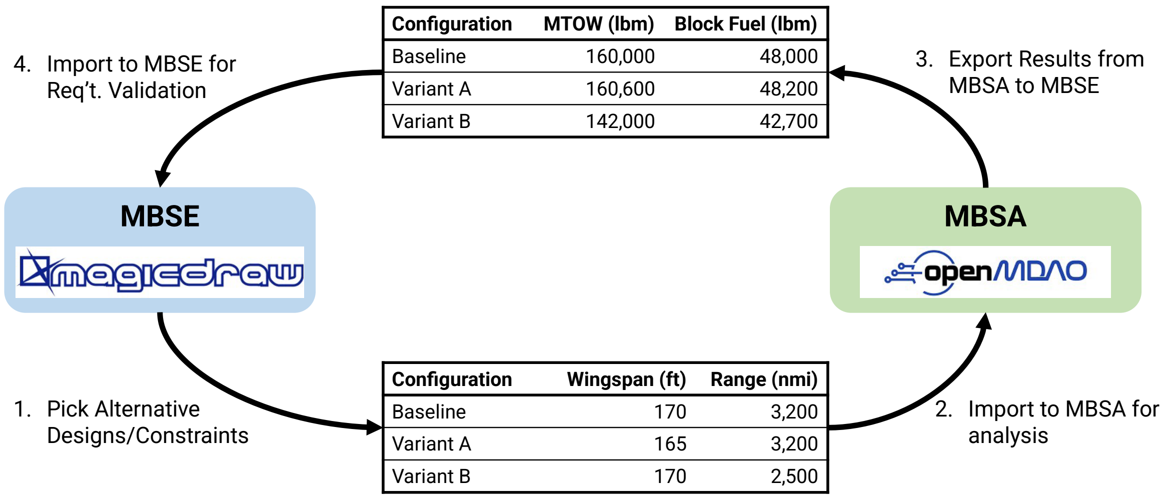
Figure 29 Requirement Derivation at Component Level

MBSE-MBSA Coupling Software Development

Motivations and Objectives

Model Based Systems Engineering (MBSE) and Model Based Systems Analysis (MBSA) tools each serve a unique purpose. MBSE tools excel at representing the system architecture in an understandable manner, capture design requirements, and easily trace design decisions made throughout the product development process[3,4]. MBSA tools excel at analyzing specific system architectures, particularly for design optimization and physics-based modeling.

In the past, industry has transferred data between MBSE and MBSA tools using CSV files or data tables. A typical data transfer workflow is shown in Figure 30, demonstrating a workflow between MagicDraw (MBSE) and OpenMDAO (MBSA). In this simple example, only two variables changed, and three configurations are analyzed. However, in larger design problems, dozens of variables may be changed, and a multitude of configurations are analyzed. If data were to be passed by CSV file for larger problems, the process would become cumbersome and increases the chances that the designer makes errors while translating between tools.


4. Import to MBSE for Req't. Validation
3. Export Results from MBSA to MBSE
MBSE
MBSA
1. Pick Alternative Designs/Constraints
2. Import to MBSA for analysis

Configuration	MTOW (lbm)	Block Fuel (lbm)
Baseline	160,000	48,000
Variant A	160,600	48,200
Variant B	142,000	42,700

Configuration	Wingspan (ft)	Range (nmi)
Baseline	170	3,200
Variant A	165	3,200
Variant B	170	2,500

Figure 30 Using CSV to pass data between MBSE and MBSA tools.

Instead, it would be simpler to automate the data transfer process such that the designer only needs to import/export data files between the tools. Performing this process automatically can reduce error while translating between design tools and can easily accommodate any problem size. Incorporating the ADH developed in CLIN 001, Figure 31 illustrates an improved workflow that the system engineer may use to develop a product. First, an ADH is created and imported into MagicDraw. Then, design decisions can be made while modifying the system architecture accordingly. After that, the system engineer can export the system model for further analysis. This process can be easily repeated for each design phase.

[3] Habermehl, C., Höpfner, G., Berroth, J., Neumann, S., & Jacobs, G. (2022). Optimization workflows for linking model-based systems engineering (MBSE) and multidisciplinary analysis and optimization (MDAO). *Applied Sciences*, *12*(11), 5316.

[4] Jeyaraj, A. K., Tabesh, N., & Liscouet-Hanke, S. (2021). Connecting Model-based Systems Engineering and Multidisciplinary Design Analysis and Optimization for Aircraft Systems Architecting. In *AIAA AVIATION 2021 FORUM* (p. 3077).

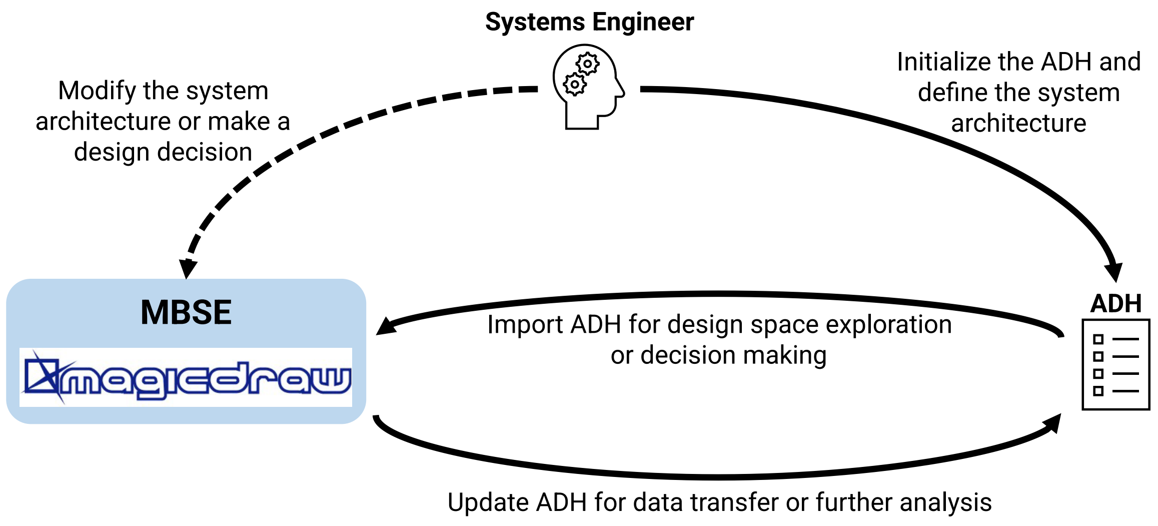

Systems Engineer
Modify the system architecture or make a design decision
Initialize the ADH and define the system architecture
MBSE
magicdraw
ADH
Import ADH for design space exploration or decision making
Update ADH for data transfer or further analysis

Figure 31 Using ADH to pass data between MBSE and MBSA tools

In the United States, little work has been done to combine MBSE and MBSA capabilities under a unified framework. However, the European Union has already begun to make advances in this area with their AGILE 3.0 and 4.0 projects[5]. Significant progress is needed to remain current with the rest of the world's design capabilities while also innovating beyond the existing state-of-the-art methodologies.

This work involves developing an application programming interface (API) between Magic Systems of Systems Architect (referred to as "MagicDraw" from herein) to easily connect MBSE and MBSA tools. The API consists of five Jython scripts to be installed as "plugins" in MagicDraw and feature their own unique capabilities:

1. *ReadADH*: read a JSON file and create a SysML model in MagicDraw.
2. *WriteADH*: export the SysML model in MagicDraw to a JSON file.
3. *UpdateADH*: compare the SysML model with a JSON file, overwriting anything in the system model that conflicts with the JSON file.
4. *ImportStereotypes*: read a JSON file and create a stereotype profile for all components with a Work Breakdown Structure (WBS) Number[6].
5. *WriteInstance*: export an Instance Specification in MagicDraw to a JSON file.

Technical Approach

The functions developed in the API are part of a larger workflow in the overall design process. Before reviewing the functions individually, Figure 32 illustrates how the API is used in the context of product design.

[5] Bussemaker, J., Boggero, L., & Nagel, B. (2023, July). The AGILE 4.0 Project: MBSE to Support Cyber-Physical Collaborative Aircraft Development. In *INCOSE International Symposium* (Vol. 33, No. 1, pp. 163-182).
[6] Department of Defense. (2022, May). Work Breakdown Structures for Defense Materiel Items. [PDF].

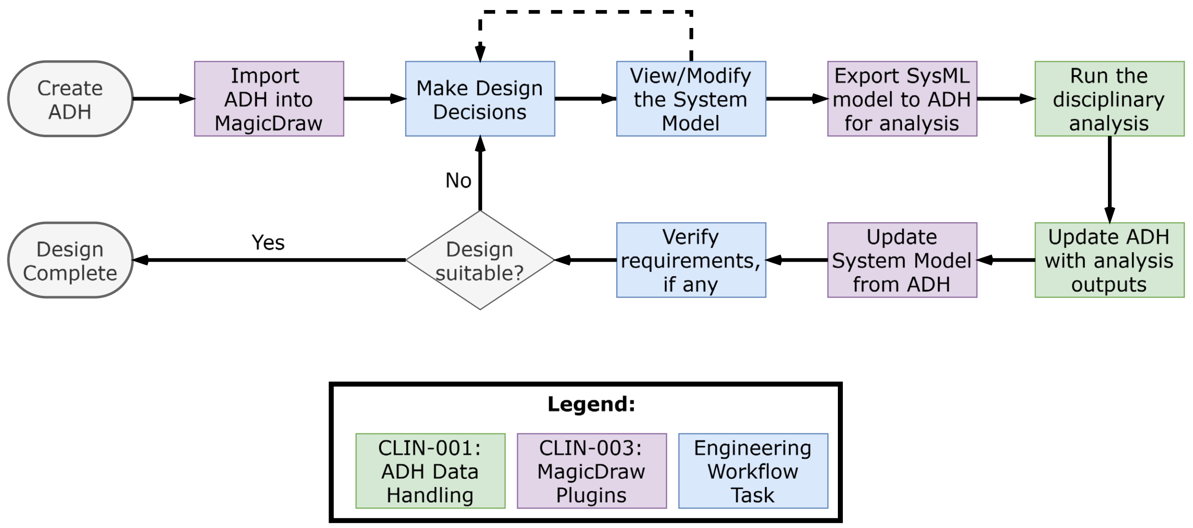
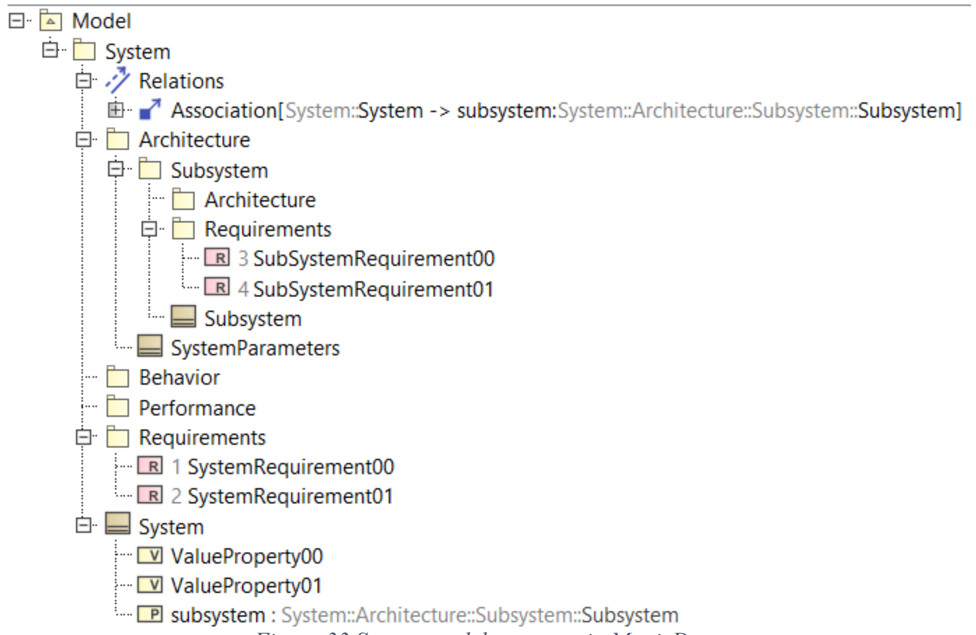

Create ADH
Import ADH into MagicDraw
Make Design Decisions
View/Modify the System Model
Export SysML model to ADH for analysis
Run the disciplinary analysis
Update ADH with analysis outputs
Update System Model from ADH
Verify requirements, if any
Design suitable?
No
Yes
Design Complete
Legend:
CLIN-001: ADH Data Handling
CLIN-003: MagicDraw Plugins
Engineering Workflow Task

Figure 32 API functions in the context of an engineering workflow

The first step in the engineering workflow is to create an ADH to represent the initial system architecture. A SysML model is generated in MagicDraw using the *ReadADH* function to parse the JSON file and create the respective model elements. Once the system architecture is in MagicDraw, the systems architect may make design decisions and modify the system model iteratively. Once this phase is complete, the system model is exported from MagicDraw to a JSON file using the *WriteADH* function. Then, the information in the JSON file is used to create an ADH, run an analysis, and create a new ADH with information (results, run settings, etc.). This new ADH is used to update the system model using the *UpdateADH* function. Once the system model is updated, the systems architect may verify any requirements, assess if the design is suitable, and then iterate on the design, if necessary.

To use these functions, the system model describing the configuration must be arranged in a structured format to accommodate interactions between MagicDraw and the ADH. Figure 33 illustrates the hierarchical structure of the system model that must be followed to use the API.



```
Model
  System
    Relations
      Association[System::System -> subsystem:System::Architecture::Subsystem::Subsystem]
    Architecture
      Subsystem
        Architecture
        Requirements
          3 SubSystemRequirement00
          4 SubSystemRequirement01
        Subsystem
      SystemParameters
    Behavior
    Performance
    Requirements
      1 SystemRequirement00
      2 SystemRequirement01
    System
      ValueProperty00
      ValueProperty01
      subsystem : System::Architecture::Subsystem::Subsystem
```

Figure 33 System model structure in MagicDraw

For each element in the ADH with a WBS Number (has “wbs_no” as the key), a package and block are made using the “name” key-value pair from the ADH. Then, within the package, up to four additional packages are made dynamically based on the ADH contents: Architecture, Behavior, Performance, and Requirements. These packages store all structured data (dictionaries) for a given system/component. However, any singular name-value pairs are created as Value Properties under the block (shown by “ValueProperty00” and “ValueProperty01”).

The Architecture package encompasses all sub-components and design parameters associated with a given component and is created if there is structured data within that level of the ADH. In other words, “Architecture” does not need to be a key specified in the ADH. For any elements residing in the Architecture package, a Part Property relationship is established between itself and the component one level above it (given by the “Association” relation). Additionally, any structured data (another dictionary) attributed to the component is also placed in the Architecture package as a block (such as “SystemParameters”).

The Requirements package contains all of the requirements associated with a given system/component. It is created if there is a “requirements” key in the ADH. Currently, requirements are created by concatenating multiple name-value pairs from the ADH. Table 1 lists the required name-value pairs that must be included in the JSON string and the final requirement text shown in MagicDraw. There is also an option to include a single key-value pair (with the key name “text”) to input a requirement as a single text string.

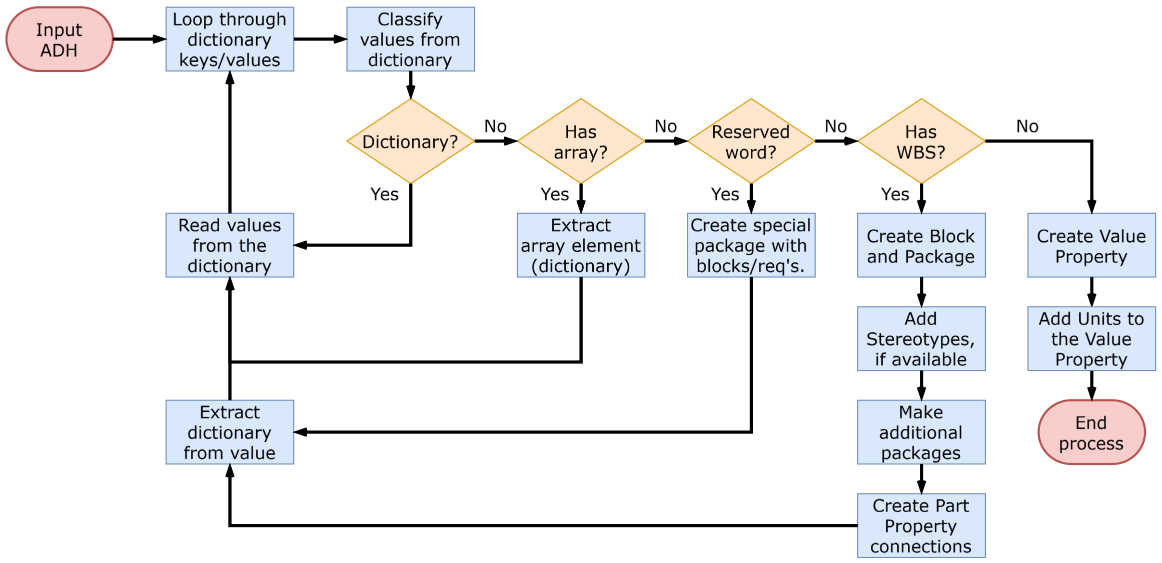
Table 1 Required name-value pairs in the JSON string and final requirement text

Key	**Value**
name	ReqName
description	ReqDesc
value.value	xxx
value.units	uuu
Requirement Text in MagicDraw	(ReqName): ReqDesc shall be xxx uuu

The Behavior and Performance packages contain information about each component, as briefly described in the CLIN 001 report. Further information about these packages is outside the scope of this CLIN report.
To ensure that the ADH is compatible with the API functions, the highest level of the JSON file must contain only one key-value pair. This is to ensure that all model elements created/updated in MagicDraw are contained within one overarching package.

With the system model structure known, each API function is introduced.

<u>*READ ADH*</u>
The *ReadADH* function parses a JSON file and creates the appropriate model elements in MagicDraw. Each level of the JSON file represents a set of systems or components that are decomposed into further parts. The computational procedure to create the system model is illustrated in Figure 34.



Input ADH
Loop through dictionary keys/values
Classify values from dictionary
Dictionary?
No
Yes
Has array?
No
Yes
Reserved word?
No
Yes
Has WBS?
No
Yes
Read values from the dictionary
Extract array element (dictionary)
Create special package with blocks/req's.
Create Block and Package
Create Value Property
Add Stereotypes, if available
Add Units to the Value Property
Extract dictionary from value
Make additional packages
End process
Create Part Property connections

Figure 34 ReadADH computational procedure

In the *ReadADH* function, a user inputs a JSON file with the system architecture to be represented in MagicDraw. Then, each key-value pair is run through the following search criteria to determine if the:

1. Value contains a WBS number
2. Value is a dictionary
3. Value is an array
4. Key contains a reserved word (Requirements, Behavior, or Performance)

If the value contains a WBS Number, then a package-block pair is made, and the appropriate Part Properties are initialized. If a stereotype profile was imported prior to calling the *ReadADH* function, then each block created will be stereotyped with its name. Then, the data structure is checked to determine if additional packages (Architecture, Requirements, Behavior, and Performance) must be made. Then, the remaining data is processed recursively by looping through the name-value pairs in the next level of the JSON string.

If the value is a dictionary and does not contain a WBS number, it represents structured data (shown as "SystemParameters" in Figure 33) and is stored as a block in the respective Architecture package.

If the value is an array, then each element is extracted and analyzed separately. One limitation of MagicDraw is that arrays are not supported. Therefore, each array element must be named separately. Within the *ReadADH* function, there is a helper function that parses the shape of the array and creates unique variable names by appending the index of each element to the name of the array. For example, a 2-by-3 array with the key "MyArray" is represented in MagicDraw with six different model elements, as listed in Table 2.

Table 2 Model element names for each array element

MyArray__0__0	MyArray__0__1	MyArray__0__2
MyArray__1__0	MyArray__1__1	MyArray__1__2

A double underscore is used to separate the array indices. An unlimited number of array indices may be included in the model element name. To be consistent with Jython, array indices are zero-biased.

If the key contains a reserved word (Behavior, Performance, or Requirements), then the data is stored within a separate package as shown in Figure 33. Examples of these include the "SystemRequirement" and "SubsystemRequirement" elements.

WRITE ADH

The *WriteADH* function parses the system model and creates a JSON string based on the blocks, packages, requirements, and Value Properties. The JSON string output includes data from the selected model element and all of its children ("owned elements"). Figure 35 illustrates the computational procedure to export the system model to a JSON string.

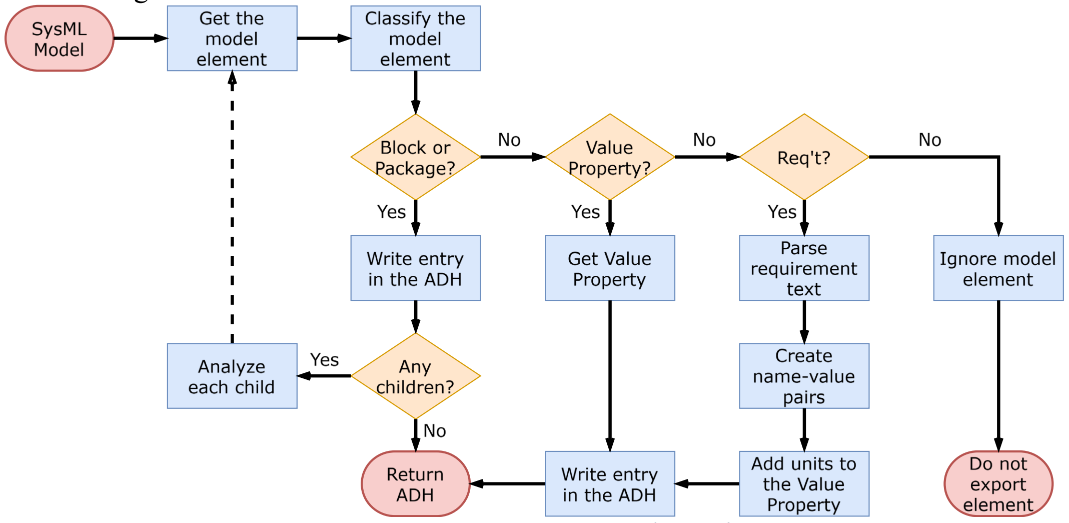


Figure 35 WriteADH computational procedure

The highest-level model element is retrieved and classified based on its element type. If it is a block or package, then an entry in the ADH is written for that component. If the model element has any children (owns another element), the function is called recursively to analyze the children further.

If the model element is a Value Property, then its value is retrieved from the system model. Its name and value are written to the JSON string as a name-value pair.

If the model element is a requirement, then its text is parsed and divided into the respective name-value pairs as previously described in Table 1. If the requirement text does not follow the previously mentioned format, the requirement text is returned as a single string in the JSON file.

If the model element does not match any of the aforementioned model elements, it is ignored and not included in the JSON string. This ensures that the system architect can include other artifacts in the system model (Requirement Tables, Block Diagrams, Instance Tables, etc.) and leverage the capabilities within MBSE without losing the ability to transfer data for MBSA.

UPDATE ADH

The *UpdateADH* function compares the values within a JSON string to the values already stored in the system model. For any values that are not equal, data from the JSON string is used to overwrite the value in the system model. Figure 36 illustrates the computational procedure for updating a system model with an ADH.

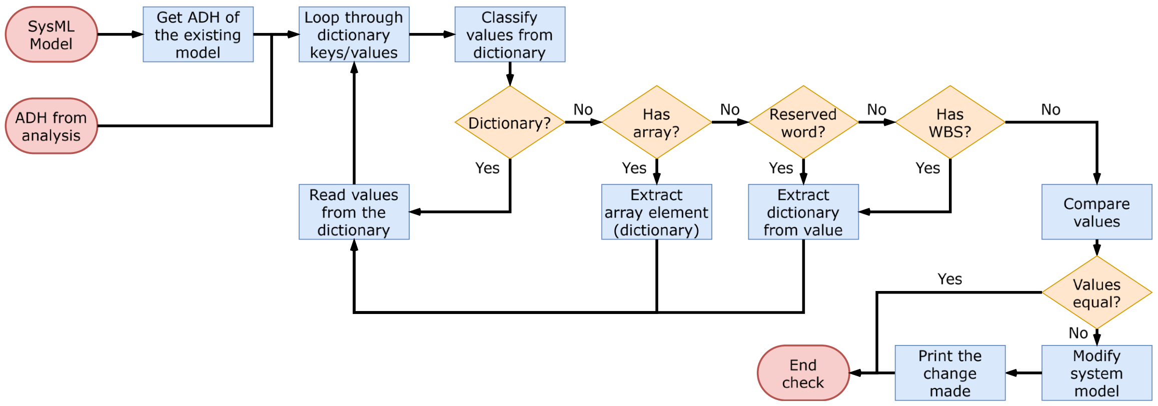

SysML Model
ADH from analysis
Get ADH of the existing model
Loop through dictionary keys/values
Classify values from dictionary
Dictionary?
No
Has array?
No
Reserved word?
No
Has WBS?
No
Yes
Yes
Yes
Yes
Read values from the dictionary
Extract array element (dictionary)
Extract dictionary from value
Compare values
Values equal?
Yes
No
End check
Print the change made
Modify system model

Figure 36 UpdateADH computational procedure

First, the code traverses all branches of the ADH provided to find all Value Properties. This is done by performing the four checks previously outlined while discussing the *ReadADH* function. If any of these checks are satisfied, the values (dictionaries) are extracted and further processed until the Value Properties are reached. Once a Value Property is found, its value in the ADH and the system model are compared. If the two values are not equal, then the value in the system model is overwritten by the one provided in the ADH. The Value Property change is also recorded in a separate text file that the systems engineer can access to understand which model elements were updated. The text file contains the qualified name of the model element and its previous and current values.

IMPORT STEREOTYPES

Aside from importing the system model, the system engineer may want to apply stereotypes to each component to better understand their purpose or function in the system. However, the systems engineer should not have to create stereotypes for each component in the system model manually. The overall workflow is shown in Figure 37.

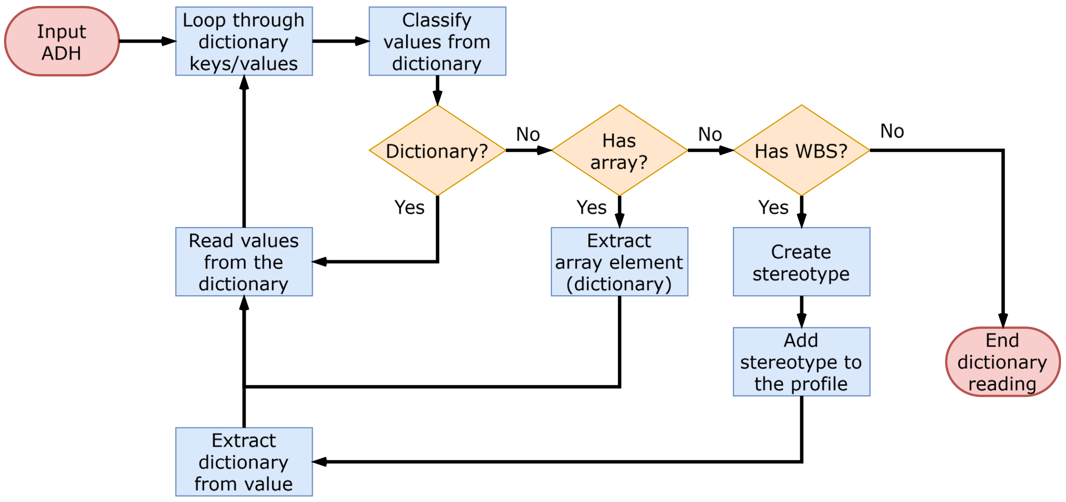

Input ADH
Loop through dictionary keys/values
Classify values from dictionary
Dictionary?
No
Has array?
No
Has WBS?
No
Yes
Yes
Yes
Read values from the dictionary
Extract array element (dictionary)
Create stereotype
Add stereotype to the profile
End dictionary reading
Extract dictionary from value

Figure 37 ImportStereotypes computational procedure

The workflow in the *ImportStereotypes* function solely parses the JSON file to identify all of the WBS Numbers and creates a stereotype matching the name of the component it corresponds to. This is accomplished by extracting all of the values from the JSON file, exploring the dictionaries recursively until WBS Numbers are reached, and then creating stereotypes for the respective components.

All of the stereotypes are stored in the system model with the name "ImportADHProfile" and can be renamed after importing. In the future, it may be useful for allowing users to name their stereotype profile a priori, particularly if multiple disparate system architectures are imported into the system model.

WRITE INSTANCE
Instance Specifications are used to describe the same system architecture with different design parameters. These model elements are particularly useful for analyzing multiple configurations and verifying requirements. Instance Specifications use slots to hold Part and Value Properties, which differs from other model elements. Therefore, a new function is required for Instance Properties to be exported from MagicDraw to the ADH. The *WriteInstance* function writes an Instance Specification and its Part and Value Properties to a JSON file. Rather than interacting with the model elements that were imported, this function interacts with an Instance Specification that is created from the imported model elements. The workflow is illustrated in Figure 38.

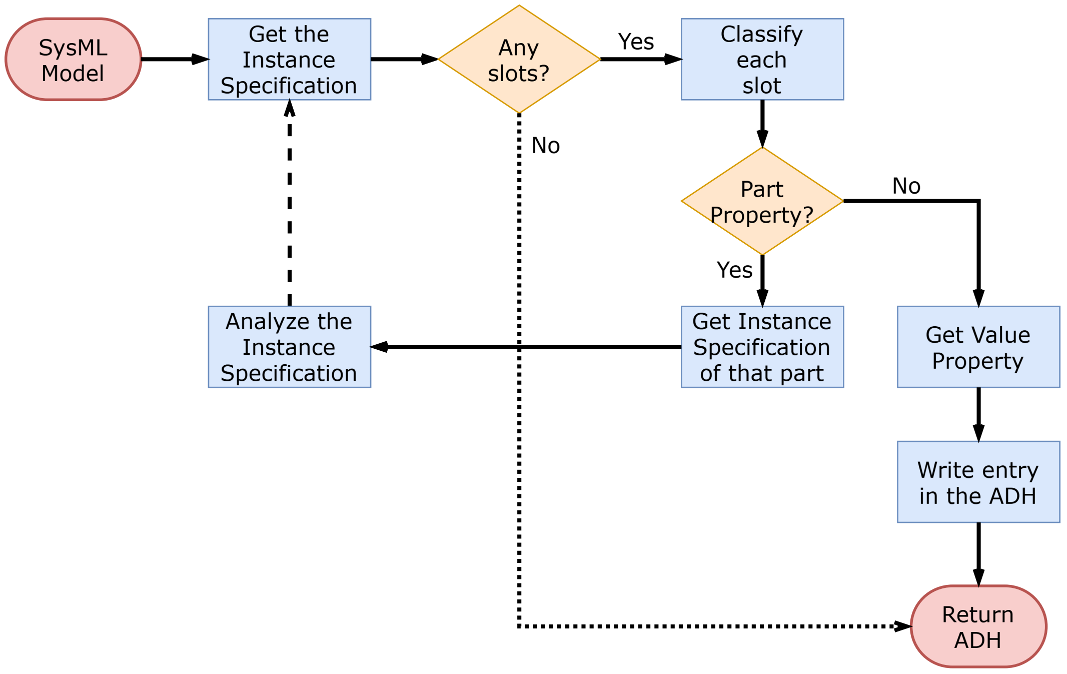


Figure 38 WriteInstance computational procedure

The *WriteInstance* function identifies the Instance Specification's Part and Value Properties and then writes the information to the ADH. If the slot contains a Part Property, then the Instance Specification corresponding to that Part Property is analyzed. If the slot contains a Value Property, its entry is written into the ADH. This function will return an ADH with the same hierarchical structure as an ADH written from the *WriteADH* function.

Demonstrations

To supplement the technical approach developed, a demonstration of the above capabilities was developed and included in the "Demo" folder stored in the GitHub repository[7]. The demonstration shows an example engineering workflow and how the systems engineer and disciplinary analysts may interact with the ADH. The steps in the workflow are to:

1. Create an empty system model
2. Import stereotypes from the ADH into a Stereotype Profile (*ImportStereotypes*)
3. Create model elements based on the ADH contents (*ReadADH*)
4. Add a component to the system model
5. Export the updated system model for analysis (*WriteADH*)
6. Update the system model after running an analysis (*UpdateADH*)

An additional demonstration for exporting an Instance Specification from MagicDraw to an ADH (using *WriteInstance*) is also included in the repository. However, this demonstration is separate from the engineering

[7] https://github.com/ideas-um/MBSAE-API

workflow and uses a simplified ADH because the academic version of MagicDraw used at the University of Michigan does not support running simulations and automatically creating Instance Specifications.

Requirement Verification Demonstration

The requirement verification is performed through instances and instance table in MagicDraw. This section will briefly discuss the method and operation to perform the requirement verification, while more details can be found in other two deliverable files: (1) CLIN_003_MBSE_MBSA_Interconnectivity_Best_Practice_NASA MBSAE.docx; (2) CLIN_003_MBSE_MBSA_Interconnectivity_Demo_2_Instruction.docx, which is in the Post Final Review Demo deliverable package. To create instance, data from multiple ADH files needs to be imported first, through the usage of the plugin *ImportStereotypes* and *ReadADH:*

1. Right click the Model package and use the plugin "MBSA&E: Import Stereotypes" to read the stereotypes from the ADH file as shown in the following figures. Please use the full file path for the JSON file when requested by the plugin, which is the same for usage of all the plugins.

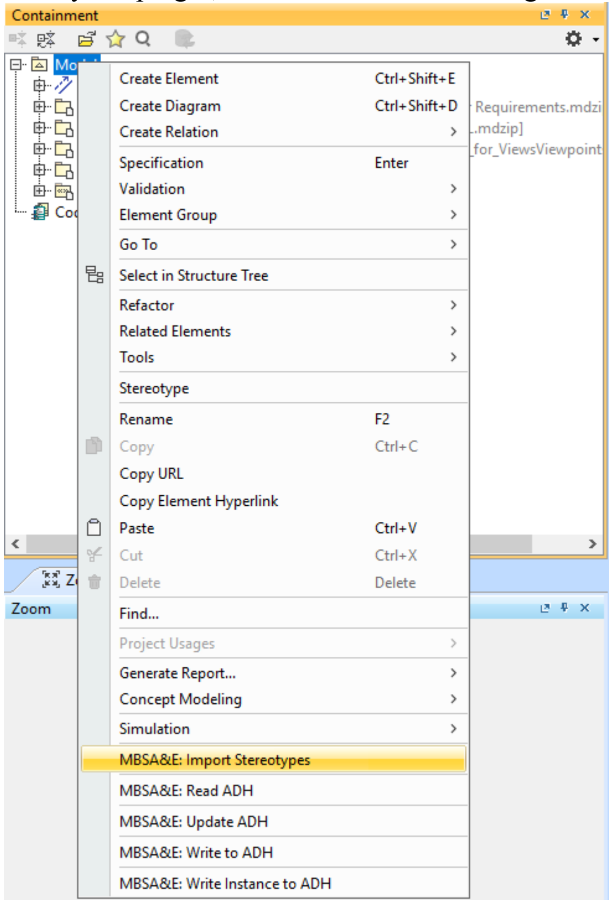
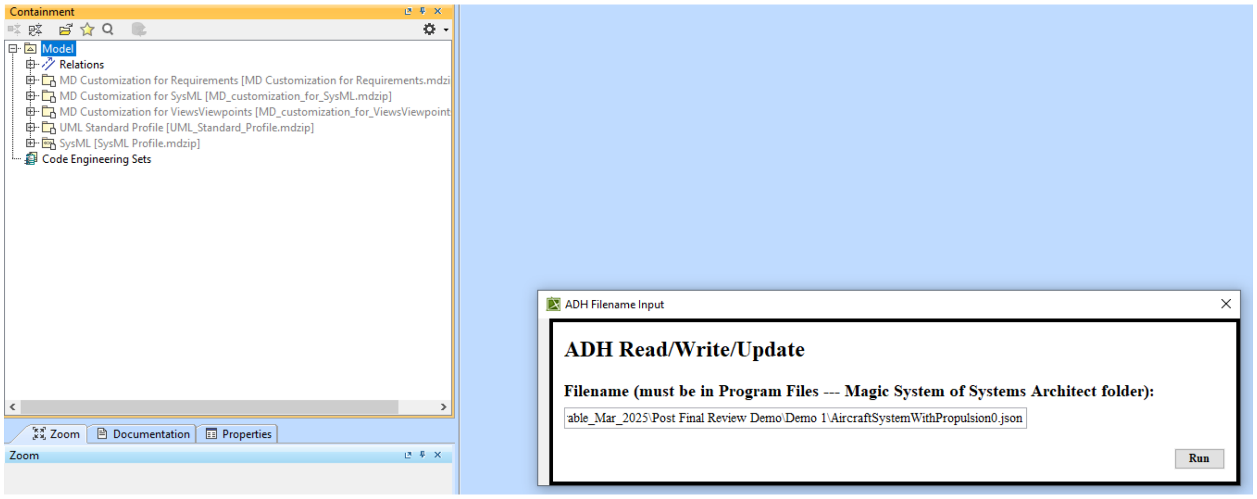


Figure 39 Execute the plugin to read stereotypes



Containment

- Model
  - Relations
  - MD Customization for Requirements [MD Customization for Requirements.mdzi
  - MD Customization for SysML [MD_customization_for_SysML.mdzip]
  - MD Customization for ViewsViewpoints [MD_customization_for_ViewsViewpoint
  - UML Standard Profile [UML_Standard_Profile.mdzip]
  - SysML [SysML Profile.mdzip]
- Code Engineering Sets

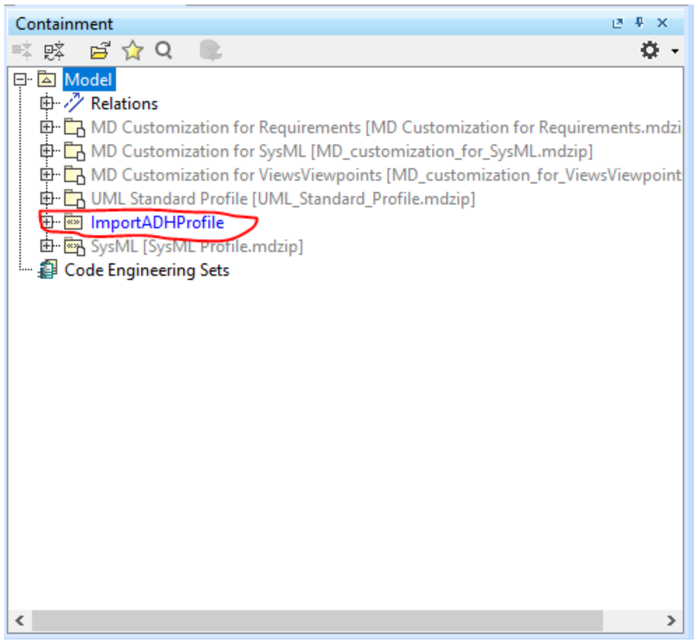
ADH Filename Input

ADH Read/Write/Update

Filename (must be in Program Files --- Magic System of Systems Architect folder):

able_Mar_2025\Post Final Review Demo\Demo 1\AircraftSystemWithPropulsion0.json

Figure 40 Specify ADH file path and run



Containment

- Model
  - Relations
  - MD Customization for Requirements [MD Customization for Requirements.mdzi
  - MD Customization for SysML [MD_customization_for_SysML.mdzip]
  - MD Customization for ViewsViewpoints [MD_customization_for_ViewsViewpoint
  - UML Standard Profile [UML_Standard_Profile.mdzip]
  - ImportADHProfile
  - SysML [SysML Profile.mdzip]
- Code Engineering Sets

Figure 41 Imported stereotype package

2. Then the model/data can be imported: right click at the Model package and use the plugin "MBSA&E: Read ADH" to import the model from the ADH file, as shown in the following figures.

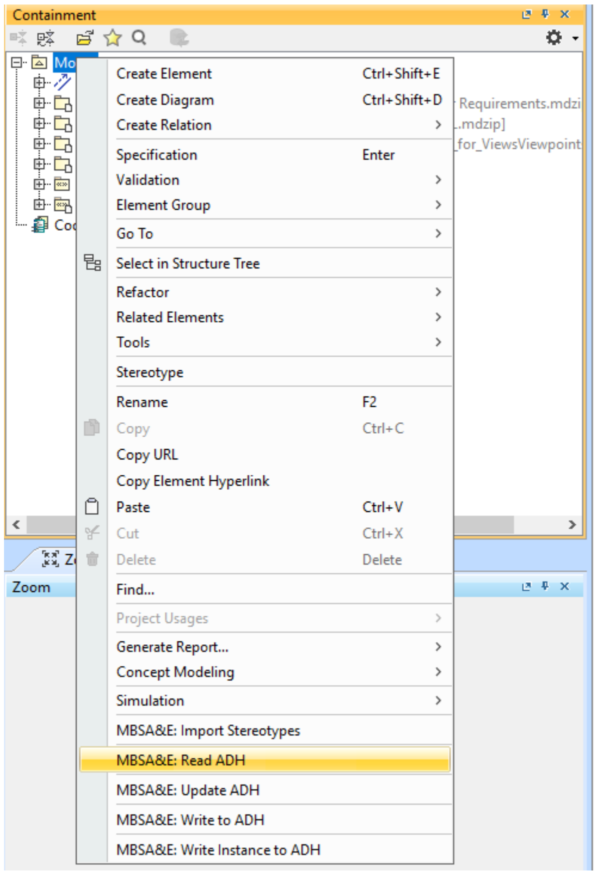
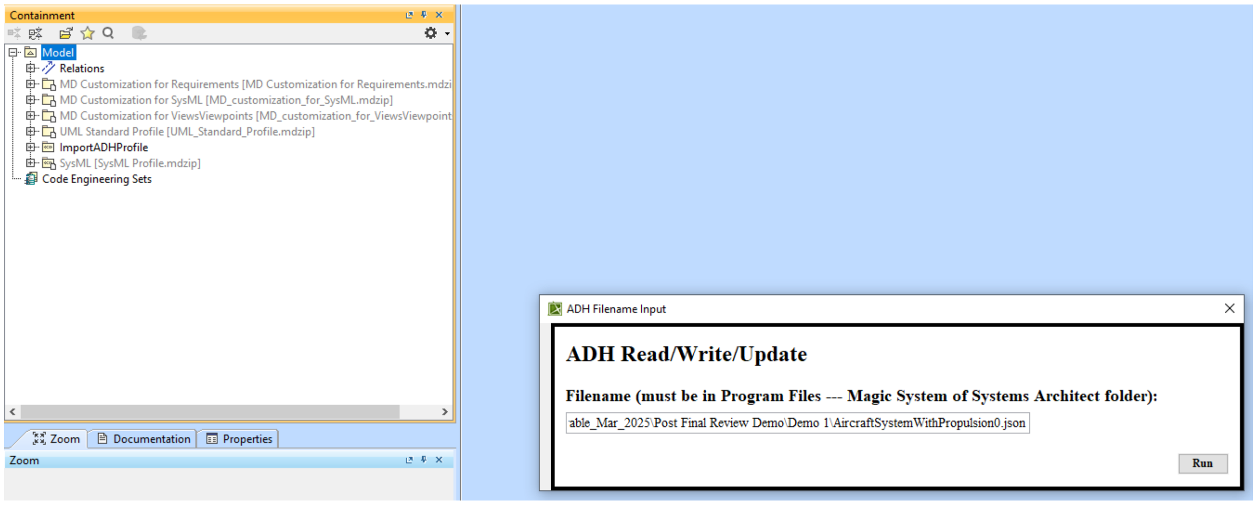


Figure 42 Execute the plugin to read ADH file


Containment
Model
Relations
MD Customization for Requirements [MD Customization for Requirements.mdzi
MD Customization for SysML [MD_customization_for_SysML.mdzip]
MD Customization for ViewsViewpoints [MD_customization_for_ViewsViewpoint
UML Standard Profile [UML_Standard_Profile.mdzip]
ImportADHProfile
SysML [SysML Profile.mdzip]
Code Engineering Sets
Zoom
Documentation
Properties
ADH Filename Input

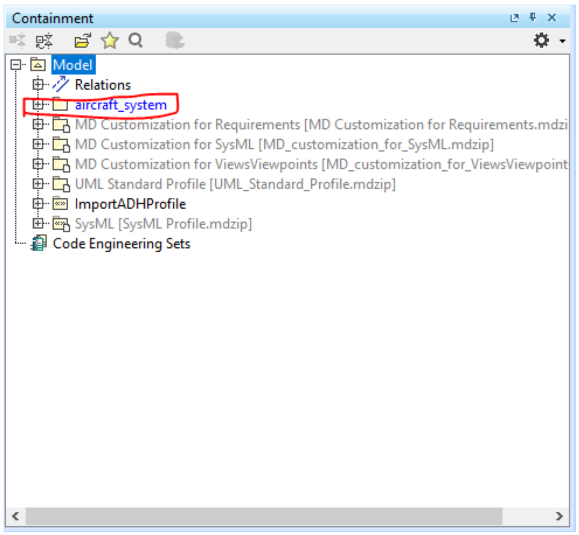
ADH Read/Write/Update

Filename (must be in Program Files --- Magic System of Systems Architect folder):

able_Mar_2025\Post Final Review Demo\Demo 1\AircraftSystemWithPropulsion0.json

Run

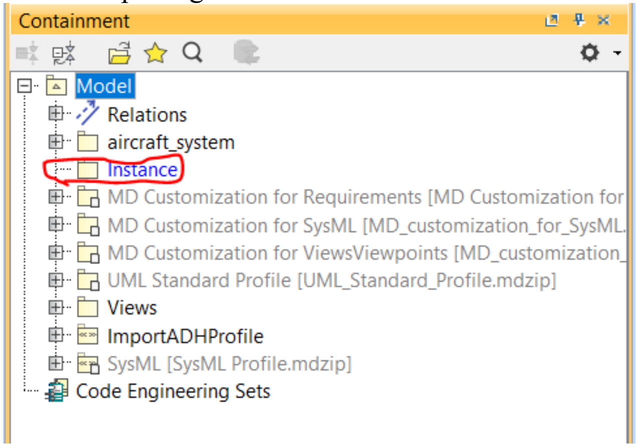
Figure 43 Specify the ADH file path and execute


Containment

Model
Relations
aircraft_system
MD Customization for Requirements [MD Customization for Requirements.mdzi
MD Customization for SysML [MD_customization_for_SysML.mdzip]
MD Customization for ViewsViewpoints [MD_customization_for_ViewsViewpoint
UML Standard Profile [UML_Standard_Profile.mdzip]
ImportADHProfile
SysML [SysML Profile.mdzip]
Code Engineering Sets

Figure 44 Imported airplane model

After the model importing, an instance package should be created to hold the instances to be saved:


Containment

Model
Relations
aircraft_system
Instance
MD Customization for Requirements [MD Customization for
MD Customization for SysML [MD_customization_for_SysML
MD Customization for ViewsViewpoints [MD_customization_
UML Standard Profile [UML_Standard_Profile.mdzip]
Views
ImportADHProfile
SysML [SysML Profile.mdzip]
Code Engineering Sets

Figure 45 Create an Instance package

Then multiple instances can be created by repeating the following process by using the MagicDraw Simulation functionality on the aircraft_system block:

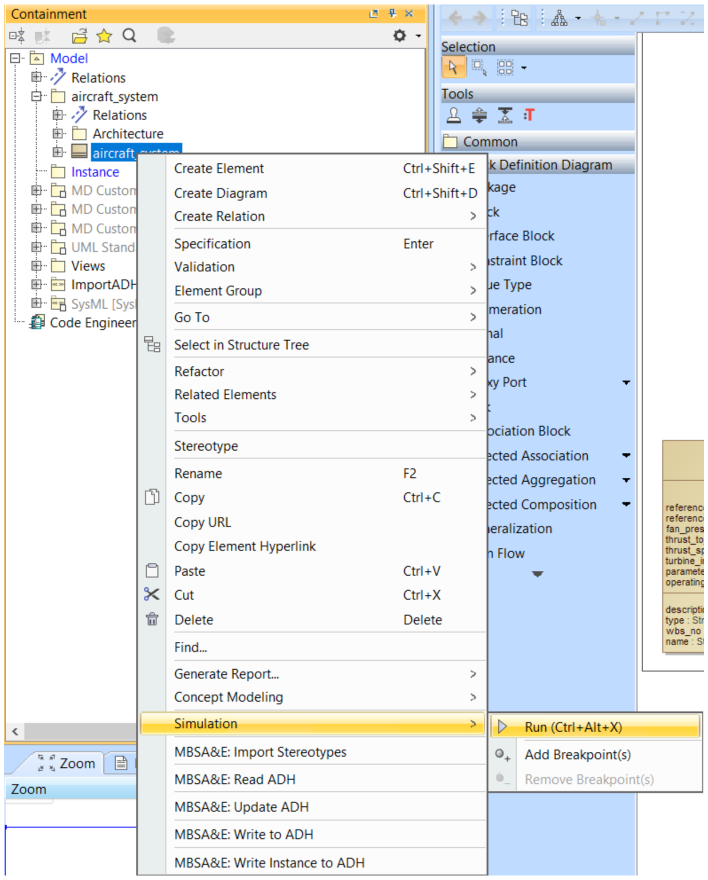
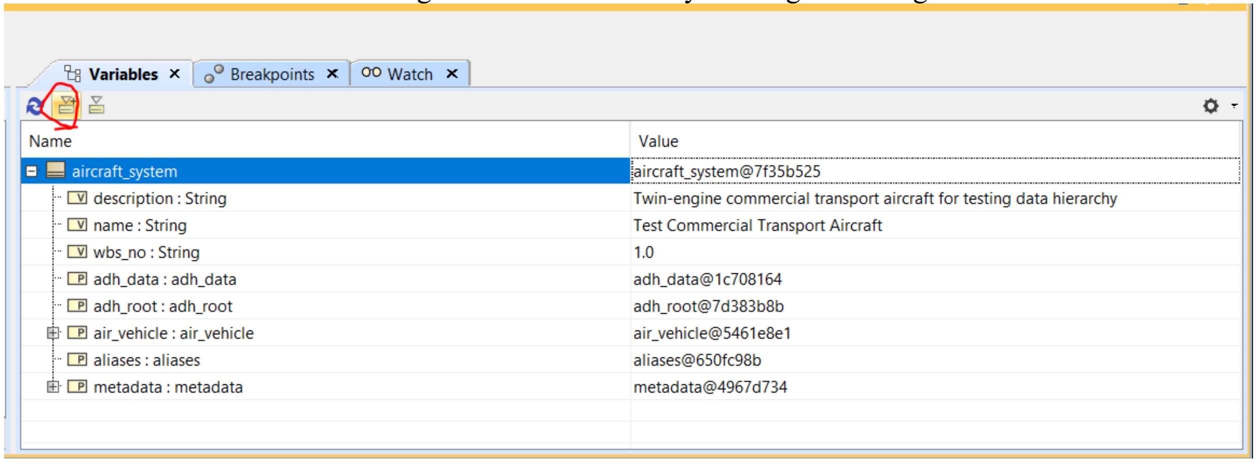


Figure 46 Run simulation to create instance

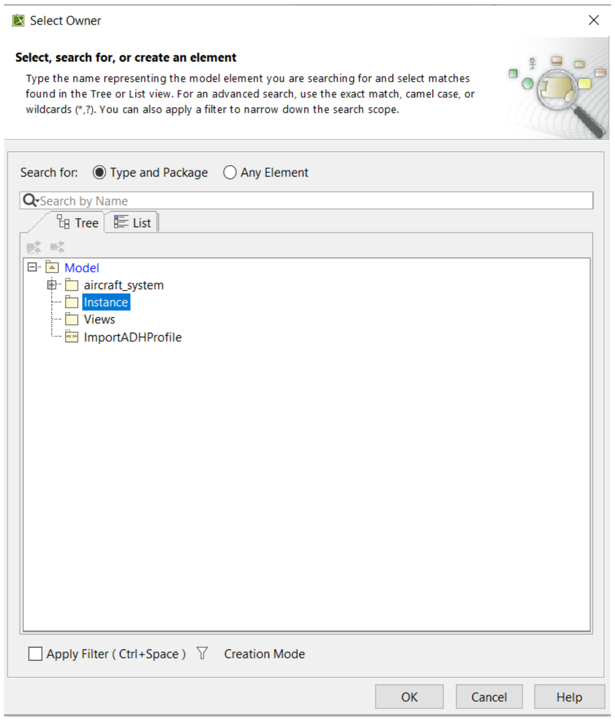
Then we can save the instance using the simulation data by clicking the saving instance button:



Name	Value
aircraft_system	aircraft_system@7f35b525
description : String	Twin-engine commercial transport aircraft for testing data hierarchy
name : String	Test Commercial Transport Aircraft
wbs_no : String	1.0
adh_data : adh_data	adh_data@1c708164
adh_root : adh_root	adh_root@7d383b8b
air_vehicle : air_vehicle	air_vehicle@5461e8e1
aliases : aliases	aliases@650fc98b
metadata : metadata	metadata@4967d734

Figure 47 Create instance using the simulation

Then we can save this instance under the Instance package we just created:


Select Owner

Select, search for, or create an element

Type the name representing the model element you are searching for and select matches found in the Tree or List view. For an advanced search, use the exact match, camel case, or wildcards (*,?). You can also apply a filter to narrow down the search scope.

Search for: [✓ Type and Package] Any Element

Search by Name

Tree | List

- Model
  - aircraft_system
  - Instance
  - Views
  - ImportADHProfile

Apply Filter (Ctrl+Space) Creation Mode

OK Cancel Help

Figure 48 Save the instance to the Instance package

To save instances from different ADH files with different values, the plugin “MBSA&E: Update ADH” is used:

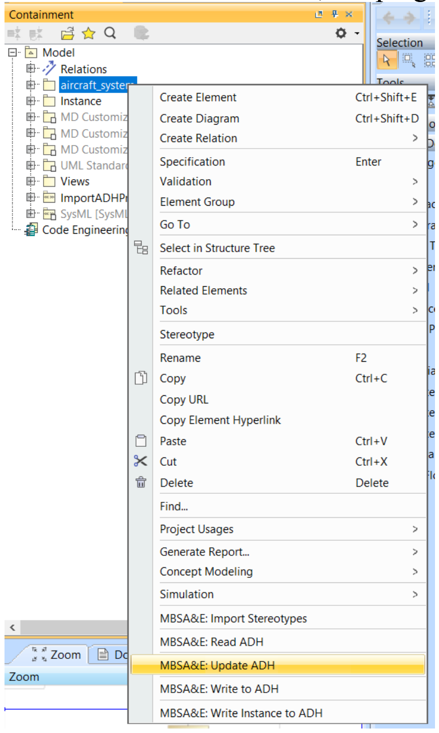
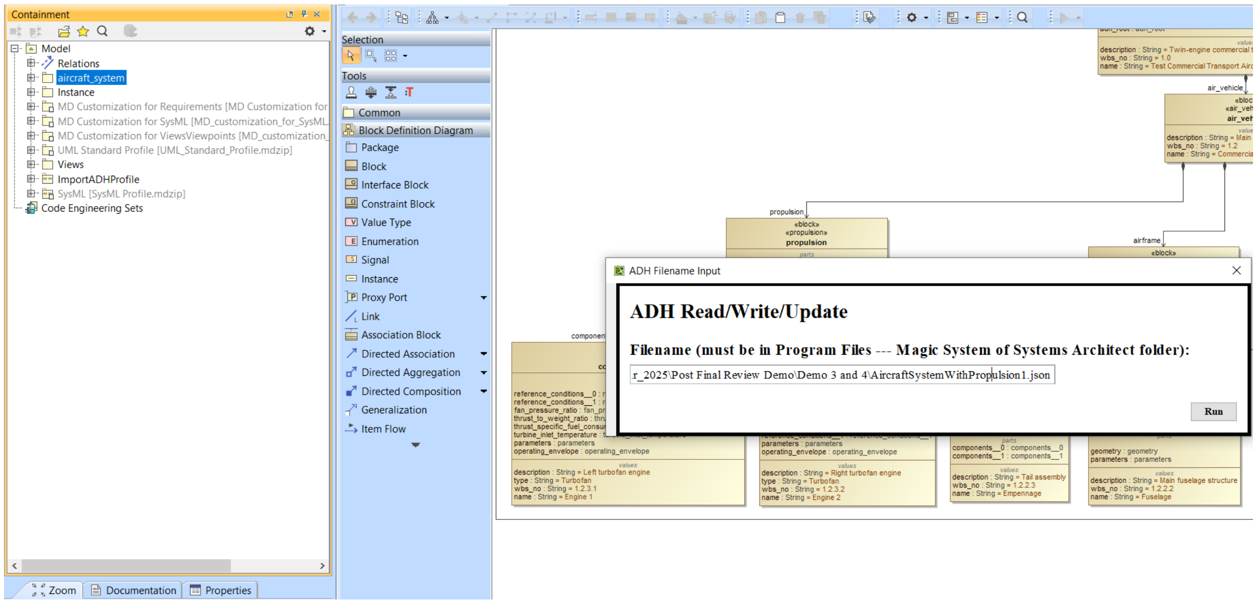


Figure 49 Execute the plugin to update the sysML model



ADH Filename Input

ADH Read/Write/Update

Filename (must be in Program Files --- Magic System of Systems Architect folder):

r_2025\Post Final Review Demo\Demo 3 and 4\AircraftSystemWithPropulsion1.json

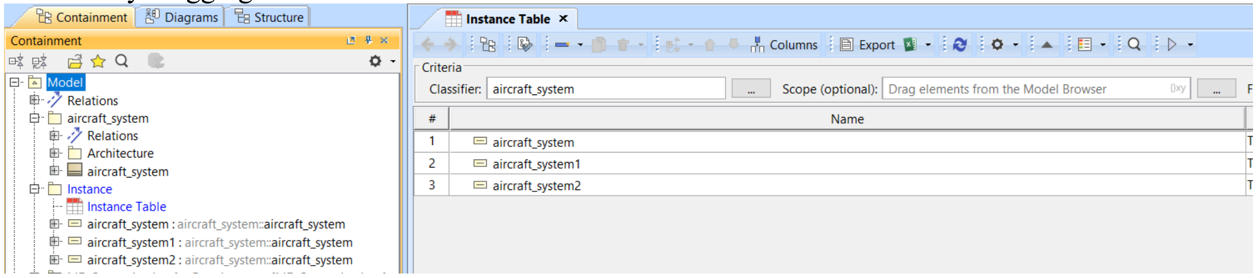
Run

Figure 50 Update the model from the new ADH file

After the model update, another instance can be saved by repeating the process shown in Figure 46 - Figure 48.

Finally, the requirement verification can be performed in the instance table:

We need to firstly create an instance table in the Instance package, and then add the different saved instances to the table by dragging them into it:



Criteria

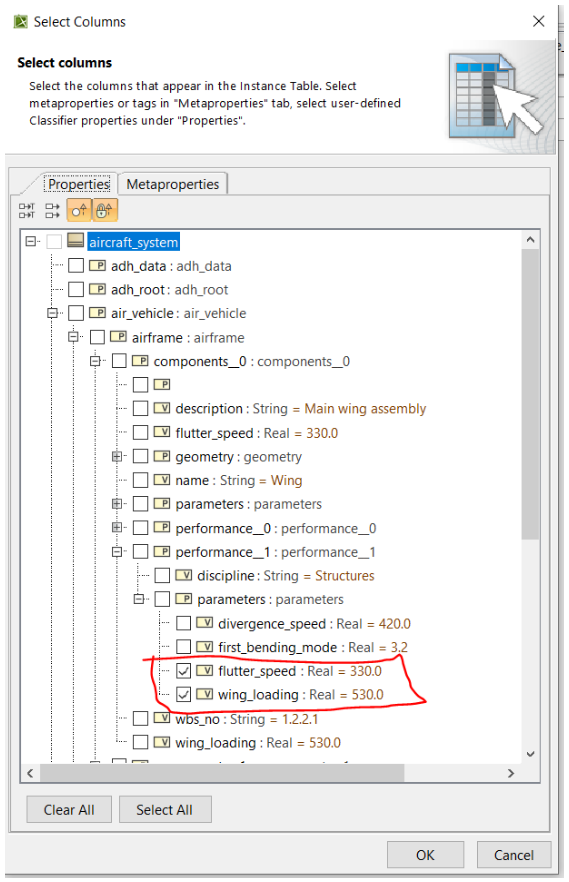
Classifier: aircraft_system

Scope (optional): Drag elements from the Model Browser

#	Name
1	aircraft_system
2	aircraft_system1
3	aircraft_system2

Figure 51 Instances in created instance table

Then we need to select only relevant columns to our verification tasks, which are the wing loading and flutter speed for this example.



Select Columns

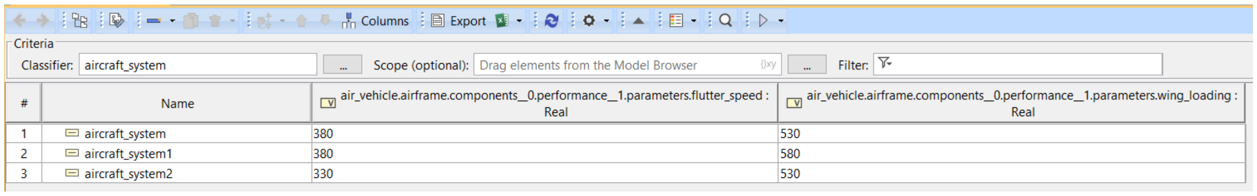
Select columns

Select the columns that appear in the Instance Table. Select metaproperties or tags in "Metaproperties" tab, select user-defined Classifier properties under "Properties".

Properties | Metaproperties

- aircraft_system
  - adh_data : adh_data
  - adh_root : adh_root
  - air_vehicle : air_vehicle
    - airframe : airframe
      - components__0 : components__0
        - description : String = Main wing assembly
        - flutter_speed : Real = 330.0
        - geometry : geometry
        - name : String = Wing
        - parameters : parameters
        - performance__0 : performance__0
        - performance__1 : performance__1
          - discipline : String = Structures
          - parameters : parameters
            - divergence_speed : Real = 420.0
            - first_bending_mode : Real = 3.2
            - ✓ flutter_speed : Real = 330.0
            - ✓ wing_loading : Real = 530.0
        - wbs_no : String = 1.2.2.1
        - wing_loading : Real = 530.0

Figure 52 Select relevant columns in Instance table



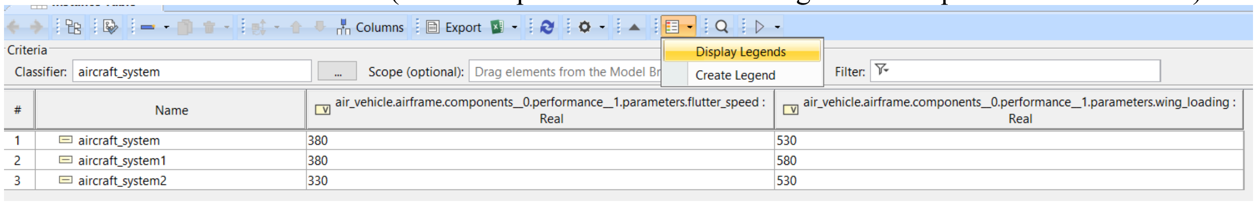
Criteria

Classifier: aircraft_system   Scope (optional): Drag elements from the Model Browser   Filter:

#	Name	air_vehicle.airframe.components__0.performance__1.parameters.flutter_speed : Real	air_vehicle.airframe.components__0.performance__1.parameters.wing_loading : Real
1	aircraft_system	380	530
2	aircraft_system1	380	580
3	aircraft_system2	330	530

Figure 53 Instance table showing instances with different values

Finally we can turn on the verification legends for the instance table, and then the corresponding verification status will be shown with colors (red for requirement violation and green for requirement satisfaction).

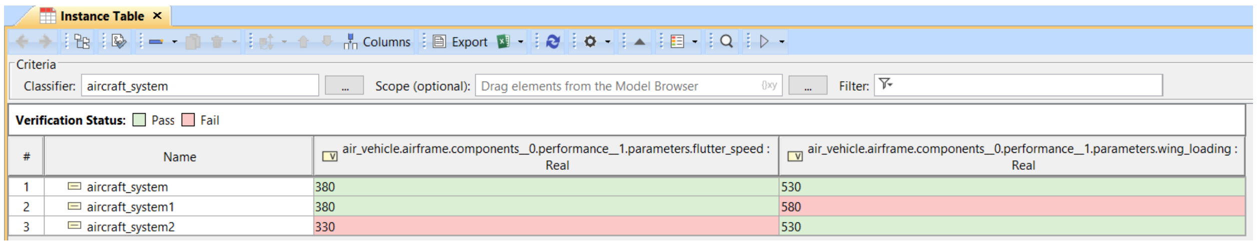


Criteria

Classifier: aircraft_system   Scope (optional): Drag elements from the Model Br   Filter:

#	Name	air_vehicle.airframe.components__0.performance__1.parameters.flutter_speed : Real	air_vehicle.airframe.components__0.performance__1.parameters.wing_loading : Real
1	aircraft_system	380	530
2	aircraft_system1	380	580
3	aircraft_system2	330	530

Figure 54 Display verification status legend



Criteria

Classifier: aircraft_system   Scope (optional): Drag elements from the Model Browser   Filter:

Verification Status: 🟩 Pass 🟥 Fail

#	Name	air_vehicle.airframe.components__0.performance__1.parameters.flutter_speed : Real	air_vehicle.airframe.components__0.performance__1.parameters.wing_loading : Real
1	aircraft_system	🟩 380	🟩 530
2	aircraft_system1	🟩 380	🟥 580
3	aircraft_system2	🟥 330	🟩 530

Figure 55 Requirement verification by instances and color

This concludes the requirement verification by using the plugins.

Conclusions

Under CLIN 003, MBSE infrastructure has been created to support the connectivity between MBSA and MBSE. The MBSE-MBSA digital connectivity has been also implemented through plugin development in MagicDraw 2022x R1 environment. On the MBSE development side, methodologies for the development of the descriptive systems engineering model, including: 1. Scope the usage of MBSE under the context of the MBSA&E program; 2. Develop the metamodel; 3. Develop the sysML profile; 4. Develop the systems engineering model; 5. Share model with collaborators; and 6. Integrate the collaborator's model back to the airplane model. The developed metamodel, sysML profile, and the airplane model provided a framework in the MBSE environment to host MBSA information. Therefore, data from disciplinary analysis can be traced to the systems engineering artifacts.

On the MBSE-MBSA digital connectivity side, the development involved developing an API within MagicDraw to allow MBSE and MBSA tools to interface with each other in engineering workflows. To accomplish this, five plugins were created to read/write JSON files and modify model elements within MagicDraw. The plugins have been publicly released to a GitHub repository. This report described the inputs/outputs for each plugin, along with coding logic driving each workflow. To highlight the capabilities that the API can perform, a demonstration was included in the GitHub repository using an example workflow.

Three additional improvements can be made to further enhance this work. First, the capability to import/export a subset of the ADH would allow a disciplinary analyst to export the part of the ADH they need, run a simulation, and update the system model with the respective results. This prevents them from accessing parts of the system model that are not pertinent to their work. Second, the ability to import an Instance Specification would help the systems engineer while analyzing multiple configurations with the same system architecture, but different design parameters. This improvement would speed up the workflow and not require the systems engineer to create an Instance Specification from the imported model elements. Third, the *UpdateADH* function currently requires that the system architecture in the system model and the provided ADH to have the exact same structure. This could pose issues if the MBSA tool outputs more information to the ADH than expected. Instead, it may be more effective to create a *DiffADH* function that lists the differences between two ADHs (similar to a version control tool), thus removing the constraint on the ADH structure. Then, the systems engineer may select which parts of the system model get updated from the new ADH.

References

1. NASA Systems Modeling Handbook for Systems Engineering, https://standards.nasa.gov/standard/NASA/NASA-HDBK-1009
2. "*NASA Systems Engineering Processes and Requirements*," NASA NPR 7123.1D, 2020
3. Habermehl, C., Höpfner, G., Berroth, J., Neumann, S., & Jacobs, G. (2022). Optimization workflows for linking model-based systems engineering (MBSE) and multidisciplinary analysis and optimization (MDAO). *Applied Sciences*, *12*(11), 5316.
4. Jeyaraj, A. K., Tabesh, N., & Liscouet-Hanke, S. (2021). Connecting Model-based Systems Engineering and Multidisciplinary Design Analysis and Optimization for Aircraft Systems Architecting. In *AIAA AVIATION 2021 FORUM* (p. 3077).
5. Bussemaker, J., Boggero, L., & Nagel, B. (2023, July). The AGILE 4.0 Project: MBSE to Support Cyber-Physical Collaborative Aircraft Development. In *INCOSE International Symposium* (Vol. 33, No. 1, pp. 163-182).
6. Department of Defense. (2022, May). Work Breakdown Structures for Defense Materiel Items. [PDF].