Demonstrating SysML v2 API Interoperability with Model Management Dashboard

AE 8900 MAV Special Problems

Presented by: Eliezer Zavala Gonzalez

Advisor: Prof. Dimitri Mavris

December 4th, 2024

Grade received: _____

Advisor's signature: _____

School of Aerospace Engineering

Georgia Institute of Technology

Honor Code Statement
I certify that I have abided by the honor code of the Georgia Institute of Technology and followed the collaboration guidelines as specified in the project description for this assignment.

Signed: _____

Demonstrating SysML v2 API Interoperability with Model Management Dashboard

Executive Summary

In the past decade, the development of various new technologies has enabled the consideration of increasingly more ambitious aerospace ventures. These come hand in hand with inherently more complex vehicle requirements and design, increasing the difficulty behind the decision-making process and, thus, selecting a solution. To support these engineering efforts, various approaches can be used to bring structure and clarity to solving these problems. Most notably, systems engineering techniques have become more popular to break down the main problem into smaller, more manageable ones for which the solutions are easier to define and attain. The collection of all lower-level solutions can then be more easily verified against the system requirements, checking whether they all achieve their intended goals.

In the past, the traditional systems engineer approach was utilized, where information was stored in static documents, and any updates or changes needed to be reflected in all connected documents. This process is not only time consuming, but it also enables the possibility of errors or absent information across these documents. Thus, extensive human-in-the-loop efforts are needed, which cannot be afforded in today's agile, fast-paced development environment. This is why the industry has turned towards the model-based systems engineering (MBSE) approach. With an emphasis on constructing an all-encompassing system model capable of automating its modification and verification activities, the MBSE approach produces a single source of truth utilized that can be utilized across all teams and disciplines.

Model-based languages were then developed to further reinforce this approach. This is where the systems modeling language (SysML) comes into play, which was specifically developed for systems engineering applications such as supporting specification, analysis, design, verification and validation efforts. Despite its integration into popular systems modeling environments, its lack of interactivity and interoperability, inability to internally perform engineering analysis, and its dependence on external tools are only a few of the imposed drawbacks that have limited its appeal. The second version of this language, SysML v2, aims to resolve these issues by providing a new robust visualization capabilities and, notably, a standardized application programming interface (API) aimed to solve model data handling, interaction, accessibility and portability.

This report aims to create a SysML v2 API Scripts library by wrapping the SysML v2 API calls in their own, single-line functions, increasing the usability, effectiveness, and thus adoption of the language and feature across disciplines and stakeholders. These functions replicate what the user would have to do in an otherwise cumbersome process using the native API json web application. Thus, this library lowers the difficulty of interacting with models housed in a server to simple function calls by leveraging the standardized API. To showcase the implementation of the library for a specific application, an interactive model management dashboard was created using the Streamlit Python library, where the user can create or select a model, and modify it by creating, updating, or deleting parts, attributes of said parts, or requirements within the model. It also shows a graphical model decomposition of the model to further improve the visualization and usability by all stakeholders.

This tool not only achieves the original language adoptions goals, but it also increases its long-term viability by virtue of using standardized capabilities.

Demonstrating SysML v2 API Interoperability with Model Management Dashboard

Eliezer Zavala Gonzalez*
ezavala19@gatech.edu

MBSE, SysML v2, API Library, Model Accessibility and Interactivity

Increasingly more complex systems are becoming prevalent in the aerospace industry. With the capability of comprehensively decomposing, and automating the analysis and verification of these complex systems, model-based systems engineering (MBSE) has become a popular approach for handling them. Model-based languages have been developed for MBSE purposes, where SysML v1 is one of the most broadly implemented into modeling platforms. That said, it poses drawbacks such as lack of easy interactivity, the dependence on external simulation packages and non-standardized APIs, and, in general, a lack of interoperability within the language itself. SysML v2 was created to not only address these issues, but also add capabilities such as, most notably, a standardized API, which objective is to solve the model interaction, accessibility, portability issues that v1 had. This report seeks to leverage the new SysML v2 capabilities to further the standardization, ease of interactivity, and interoperability of SysML v2 by creating an API Scipts Python library that replaces the web app for creating or modifying models, and implement it in a model management dashboard as a use-case example.

Nomenclature

API	Application programming interface
GET	HTTP method used to retrieve data from a server
MBSE	Model-based systems engineering
POST	HTTP method used to send data to a server
SysML	Systems modeling language
UUID	Universally unique identifier

I. Introduction

Engineering efforts can be described as top-down decision-making process. They can be generally described with follow a certain sequential process of establishing the need, defining the problem, establishing value objectives, generating feasible alternatives, evaluating the alternatives, and finalizing with making a decision.[1] This top-down decision-making process starts with decisions made with large amounts of uncertainty that, if left unaddressed, can cause unintended consequences down the line, producing expensive redesigns, backpedaling, and extending the project timeline. With increasingly more complex design problems, there is a need for methods that not only these efforts support the high-level goals, but also manages and balances numerous disciplines that form a cohesive final design. The increase in design difficulty, as well as the rapid rate at which they change, calls for an approach that can break down the main problem into many smaller problems for which the solutions are easier to define and attain, while simultaneously being capable of verifying the collection of solutions achieve the goals. This is where systems engineering comes into play, a discipline which employs methodologies that can conceptualize these complex design and integrate their deep interdisciplinary elements. This report describes the employment of the model-based systems engineering (MBSE) approach by creating an interactive model management dashboard that leverages the new features offered by the new and improved version of the leading systems modeling language (SysML).

II. Background

A. Systems Engineering

The International Council of Systems Engineering defines systems engineering as "a interdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods."[2] It brings structure to these complex systems design, increasing the growing body of knowledge about the system, such as the key design parameters and target metrics. It supports the top-down decision making process by defining and translating requirements into functions, which are then allocated into systems, subsystems, or components through a physical decomposition process, which are then integrated into various feasible and viable designs through systems synthesis. After a systems analysis, the alternatives are evaluated and an optimal systems is chosen as the final design. The system design process can be described with the Vee-model, where the left-hand side describes the decomposition and definition phase where design decisions are made, and the right-hand side describes the integration and verification phase where, as the name implies, the designed is verified against the specifications or requirements which must be met.

This left-hand side can be expanded, where a branch-off depicts a design analysis performed via a virtual model integration.[3] This has various advantages, such as improved requirements traceability at all levels of design, continuous requirement verification, and reduced risk and design life cycle cost, serving as a virtual representation of the system and transforming the development process from a sequence of discrete decisions to a continuous spiral development of one unified product.

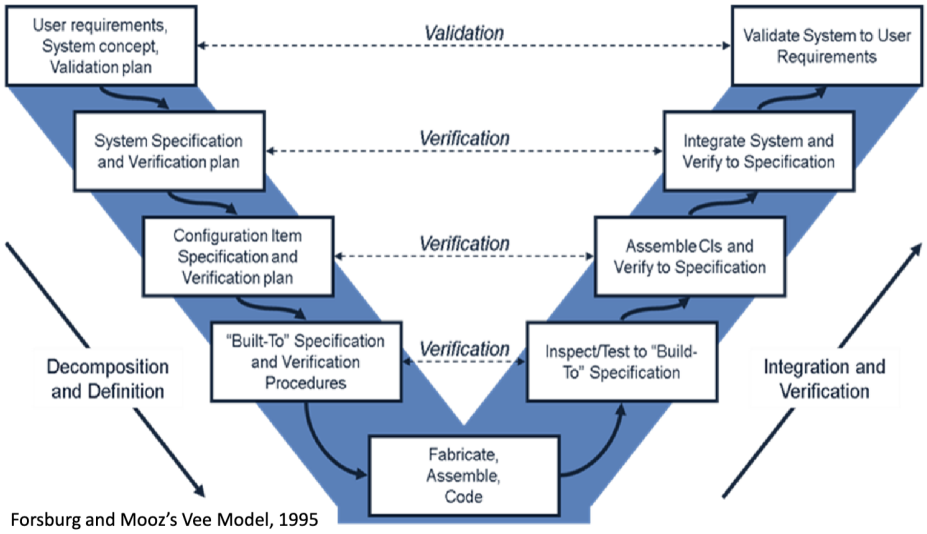

Fig. 1 Systems engineering Vee-model.

B. Model-Based Systems Engineering (MBSE)

The virtual model can be achieved with the implementation of MBSE, resulting in a model-centric process. It employs an emphasis on constructing a model that captures all aspects of the system. Additionally, it supports automatic requirements, design, analysis, verification, and validation activities, where for example changes to the requirements are instantaneously reflected throughout the model. Thus, the model can act as a single source of through across all teams and disciplines. Compared in contrast with the traditional systems engineering approach – requirements are stored in static documents where manual updates are needed, and all changes need to be changed in all relevant documents, enabling the possibility of errors or absent information and requiring extensive human in the loop efforts – it is clear that it lacks the necessary agility to address the new challenges imposed by these new, complex aerospace systems.

Originally, MBSE was created as an effort to close the gap between systems and software engineering. As the former began to consider these complex systems with increasingly more comprehensive components, a communication disconnect was found between the systems and software counterparts that hindered the development and performance of

the system. Today, it has expanded to more than just that, rather, its about employing systems engineering practices on all fronts, where MBSE has become the most vital part of the virtual model.

C. Systems Modeling Language (SysML): v1 vs. v2

MBSE languages were created to specifically represent the design of a system, better interface the disciplines, and deliver capabilities to these new complex problems. This is where the systems modeling language (SysML) comes into play: a general-purpose modeling language for systems engineering applications, which supports specification, analysis, design, verification and validation efforts of a broad range of systems and systems of systems. Despite its popularity and broad implementation in various MBSE modeling platforms, it comes with certain drawbacks that have emerged over the years. Some of these are lack of easy interactivity, the inability to perform engineering analysis internally, and the dependence on external simulation packages and non-standardized APIs.[4] In essence, it lacks interoperability within the language itself, always dependent on external tools to achieve a goal more involved than what the basic capabilities of the language can offer.

The new and version of SysML (v2) attempts to not only address these drawbacks, but also drastically improve upon the original version. It brings the kernel modeling language (kerML), a new metamodel which provides an application independent syntax and semantics for creating more specific modeling languages.[5]. It also implements a new textual view that serves as a model scripting language. This is complemented by a new robust model visualization capabilities that works hand-in-hand with the textual view, bringing updated graphical views with any script changes. Finally, the creation and implementation of a SysML v2 standardized application programming interface (API), which aims to solve the model and/or information import/export, interaction, accessibility, and portability issues that v1 had. Hence, these features solve the lack of internal interoperability and reduce the dependency on external tools by virtue of language modeling and data extraction standardization.

III. Method

This project explains the creation of a SysML v2 API Scripts Python library and a Python dashboard interface that leverages said library with the objective of increasing accessibility to, and facilitate analysis of, SysML v2 server model via the API. Given a SysML v2 model uploaded to the server, any user should be able to access said model and all its respective data solely through the dashboard, complete bypassing any prior knowledge required to handle APIs.

A. SysML v2 API Setup

In order to setup the SysML v2 API on my local computer, the instructions outlined in the *SysML v2 API Services* GitHub were followed.[6] Note that this was set up in a Macbook Pro, which means slightly different steps had to be taken for it to work on macOS. Nevertheless, the same tools outlined in the GitHub were used to setup everything, only its macOS counterpart. The following are key notes taken during the setup process regarding some steps in the GitHub page:

- Usage Scenarios: followed Scenario 2, which is used to run the API locally on my machine.
- Running PostgreSQL Docker Container: the same steps were followed to run the container. That said, it is important to note that if the container (not the image) is deleted at any point, all models or projects housed inside of that container will be deleted as well, losing them all.
- JavaJDK 11 Setup: finding a JDK 11 version that worked on my machine was troublesome at first. The version that worked can be found at the following Microsoft Learn page.[7]
- Sbt Setup: followed option 2 to run sbt as a standalone on my machine.
- Run SysML v2 API & Services: before following these steps, make sure to change the directory in the command line to the cloned "SysML-v2-API-Services" repository.

B. SysML v2 API Services

The SysML v2 API has various calls one can do via the web app, which can be seen in Figure 2. Note how these consist of either `POST` or `GET` calls, which respectively send data to or retrieve data from a server. To know which version of the project is called for, however, the user must know the `COMMIT` of interest, where the latest commit has the most up to date version of the project. Throughout this paper, SysML v2 models will also be referred to as "Projects" within the API, where the components, parts, requirements, etc. of said model will be referred to as "Elements". For version control purposes, every change done to a project will be identified by a new "Commit"; this includes creation of

elements, addition of attributes to each part, and any other change. Each project will be housed in a server with a host name called `Host`, where more than one project can be found. Therefore, the library consists of functions and a Python `Class` called `Project` whose purpose is to replicate these calls.

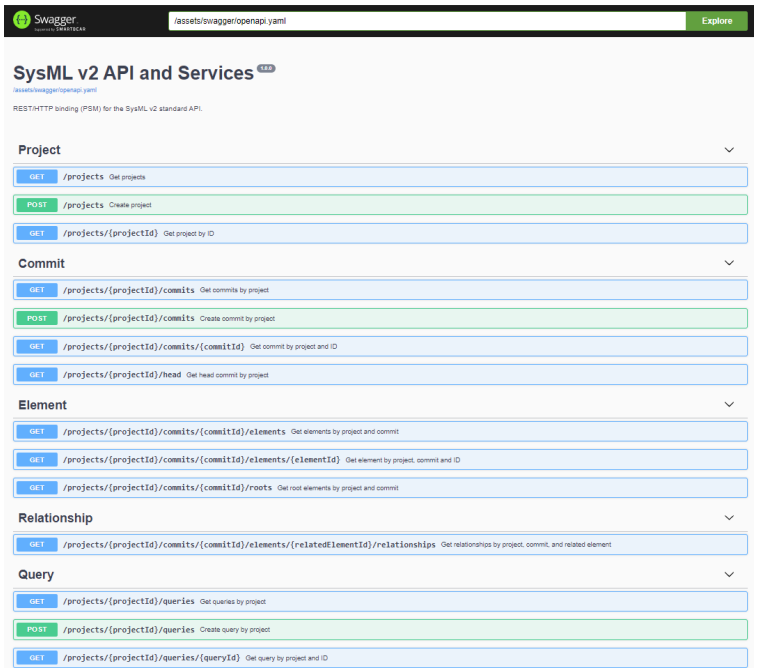

Fig. 2 Swagger web app for the SysML v2 REST/HTTP API.

C. SysML v2 API Scripts Python Library

The library is composed of a series of scripts that encapsulate the steps to create projects, parts, attributes, and requirements using the API with a single, simple function call. Fundamentally, they take some input information, create the respective commit body for a part, attribute, or requirement with the input information and appropriate meta-model class (PartUsage, AttributeUsage, RequirementUsage), and creates the commit to the selected project.[8] As these scripts are used, they not only update the model in real time, but also keep the information the script uses up to date. That is, the scripts are constantly fetching the most up to date information from the model after they are executed. The *SysML v2 API Cookbook* GitHub page was heavily used as reference to create these functions.[9]

The main piece of information needed to perform the API calls are the universally unique identifiers (UUID) of each API component. These are the project ID, commit ID of project commit of interest, and the element ID of all elements within the project. Having these allows the user to fully access, navigate, and modify the project. The following general use functions were created:

1. Host

- `change_host(new_host)`: changes the `Host` location to whatever new address the user desires
- `view_host()`: prints the current host name

2. Projects

- `projects_list()`: returns a dataFrame of all projects within the host
- `projects_names_list()`: returns a dataFrame of all project names within the host
- `projects_IDs_list()`: returns a dataFrame of all project ids within the host
- `new_project(project_name, project_description='', repeat=False)`: creates a new project in the server with the respective project name and description. If the user wants to create a new project with the same name as an already existing project, they can do so with the "Repeat=True" argument.

The user can then initialize a `Project` using the `Project(name=None, id=None, index=None) Class`, which needs either the project index within the dataFrame retrieved from the `projects_list()` function, the project name exactly how it is shown in said dataFrame, or the project ID found using the `projects_IDs_list()` function. Once the project is initialized, the following variables are automatically defined and the following methods are available for the user:

3. Project Information

- `Project.index`: assigns the index at which the initialized project is located within the projects dataFrame
- `Project.name`: assigns the name of the initialized project
- `Project.id`: assigns the ID of the initialized project
- `Project.latest_commit`: assigns the ID of the latest project commit
- `Project.current_commit`: assigns the ID of the commit the user wants to currently work with. Defaults to the `latest_commit` value when the project is initialized.
- `Project.all_elements`: dataFrame of all elements within the initialized project. Consists of a name, ID, and Type column.
- `Project.elements_names`: name column of dataFrame of all elements within the initialized project
- `Project.elements_ids`: ID column of dataFrame of all elements within the initialized project
- `Project.elements_types`: Type column of dataFrame of all elements within the initialized project
- `Project.all_reqs`: dataFrame of all requirements within the initialized project

4. Project Methods: Commits

- `Project.select_commit(self, index=None, id=None)`: assigns the ID of specified commit to the `Project.current_commit` attribute
- `Project.select_most_recent_commit(self)`: changes the value of `Project.current_commit` to the value of `Project.latest_commit`

5. Project Methods: Parts

Note: though the scripts mention creating, updating, or deleting elements, these are actually referring to creating, updating, or deleting **parts** of the model. Parts, attributes, and requirements are all considered elements of the model. This was a correction that was not able to be implemented in time.

- `Project.create_element(self, name, owner_name=None, repeat=False)`: creates a part in the model using the PartUsage class
- `Project.delete_element(self, name, id='')`: deletes the named part and all owned parts, attributes, and requirements. Note that I had trouble with this function when deleting an element that has owned elements. Ideally, try to delete elements with a bottom-up approach. If it does not work, the solution usually lied in deleting all owned elements (parts, attributes, and/or requirements) right at the beginning of the `__INIT__ TREE` section of the `__init__` function. That is, before calling any Treelib function (all Treelib functions are explained in section below)
- `Project.update_element(self, name, new_name, new_owner=None)`: updates the named element with a new name and/or a new owner

6. Project Methods: Attributes

- `Project.add_attribute(self, attribute_name, value, element_name)`: adds an attribute to the named element with specified attribute name and attribute value
- `Project.remove_attribute(self, attribute_name, id='')`: removes named attribute from its owning part
- `Project.update_attribute(self, attribute_name, new_atribute_value)`: updates the named attribute with a new attribute name and/or a new attribute value

7. Project Methods: Requirements

- `Project.create_requirement(self, req_name, description, owner_name, repeat=False)`: creates a new requirement with said requirement name, description, and under the named owner
- `Project.delete_requirement(self, req_name, id='')`: deletes named requirement
- `Project.update_requirement(self, req_name, new_req_name=None, new_desc=None)`: updates named requirement with a new name and/or a new description

D. Using Treelib to Create Model Visualization

Treelib is a Python library that allows the user to efficiently create Tree data structures, supporting operations like traversing, insertion, deletion, node moving, shallow/deep copying, subtree cutting etc.[10] Upon initializing a project with the API scripts library, there are functions that are automatically run that create a decomposition of the model as a tree data structure. Additionally, it leverages PyGraphviz – a Python interface to the Graphviz graph layout and visualization package, which allows the user to create, edit, read, write, and draw graphs using Python to access the Graphviz graph data structure and layout algorithms – to create an image of said model decomposition, which can be used for the dashboard, or, for example, sharing with stakeholders.[11]

This tree code was written as their own dedicated functions within the library that are called within the other API script functions. That is, the tree code was written to be decoupled from the rest of the library, which means the library can still be used for non-visualization purposes, or, in other words, purely model manipulation purposes. Additionally, if a programmer wants to modify the open-source code by, for example, removing the Treelib code because they do not need it, they can easily do so by just removing the lines these tree functions are called in within the API script functions.

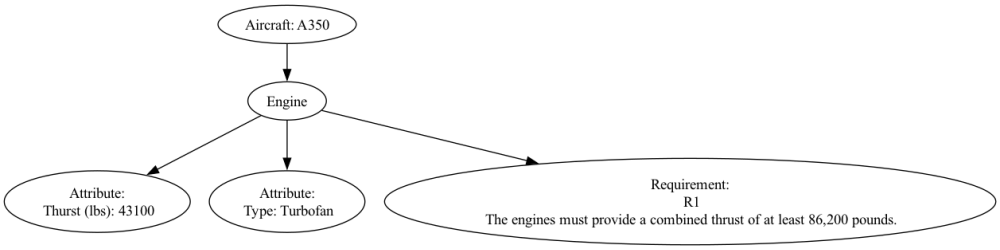

Fig. 3 Example model decomposition created with Treelib

E. Dashboard Development

The dashboard layout and application are built with *Streamlit*, a free, open-source Python library that allows users to create and share web applications.[12] It leverages the SysML v2 API Scripts Python library to seamlessly allow the user to access the SysML v2 models in an user-friendly interactive environment.

It is composed of a sidebar and a main section. The sidebar contains the Project Selection/Creation section where the user can select existing projects to view in the dashboard, or create a new project and start to manipulate it. Additionally, the sidebar contains the option of viewing the elements in a table format. These show the dataFrames of the parts, attributes, and requirements. In the main section, the model window shows the model decomposition created by the Tree code within the library. Most notably, the main section contains the element manipulation section, where the user can create, update, or delete elements within the model. A screenshot of the dashboard can be seen in Figure 4

IV. Implementation

A simple model of an A350 was created purely with the dashboard to showcase how powerful it is compared to its web app counterpart. The new project was created by simply clicking on the "Create New Project" button, naming it, and providing a description. From there, the parts of the A350, the fuselage, wings, engine, and cabin, were created via the Element Manipulation section. Attributes were then added to their respective parts, and requirements were created and connected to their respective parts, finalizing the model.

Trying to replicate this same process in the web app would involve a `POST` to create a project, finding the project ID among a list of possibly hundreds or thousands of models (depending on how many there are in the server), creating a `COMMIT` for every element that the user wants to add, each with a different commit body if its a part, attribute or

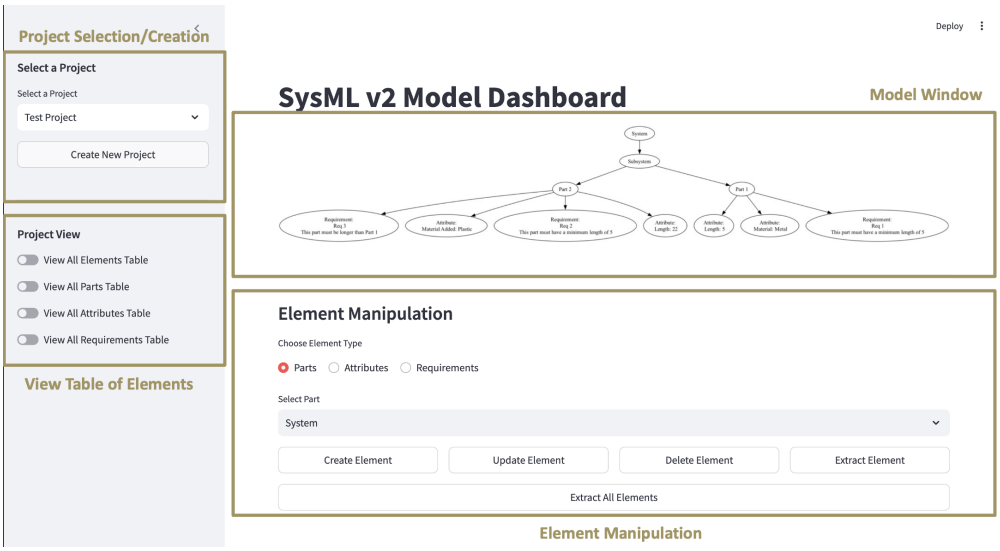

Fig. 4 SysML v2 Model Management Dashboard

requirement. A specific and precise commit body must be used or else the commit will fail. This is especially important for the attributes and requirements since the user will have to define their owners, which is a lengthy process of finding the element ID of interest (again, from a list of hundreds or thousands of elements within the model), and adding it into the precise commit body. Note that if the page is refreshed, or if the outline sections of the web app are collapse, all information that the user had inputted will be deleted, and they will revert to the default template provided by the web app.

Comparing these two approaches, it is clear that the model management dashboard is far superior in its own regard. It is a short and simple process that takes seconds compared to the cumbersome and convoluted process that is trying to create even a simple model via the web app.

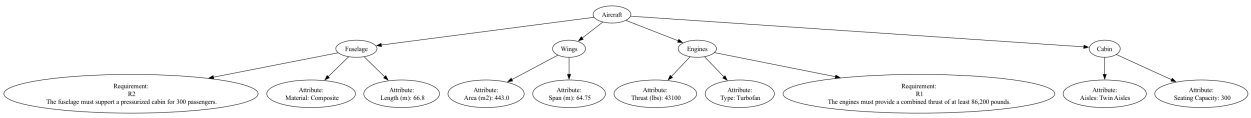

Fig. 5 A350 Model Created via the Model Management Dashboard

V. Conclusions

The API Scripts library enables the interoperability and integration of SysML v2 models with other disciplines, where the model management dashboard showcases a successful use-case of the library, highlighting its usefulness. Moreover, it completely bypasses the cumbersome process that is creating and modifying a SysML v2 model using the standard web application, reducing the time and effort required by the users. Finally, the dashboard nurtures ideas of how it can be expanded to enable more comprehensive or complex use-cases. Some examples of these may be mass-importing elements into a model, or even mass importing entire models with their respective elements, integrating SysML v2 models with third-party analysis tools, modifying the library the satisfy specific needs different users may have, and combine it with a custom dashboard that has all the features they may need or want their stakeholders to have access to. Given its open-source nature, there are numerous possibilities on how to expand this library and dashboard to whatever the user may want.

Overall, this project contributes to the development of SysML v2 by increasing the accessibility of stakeholders (programmers, project managers, general users, etc.) to SysML v2 server models via API without prior API knowledge required.

References

[1] Mavris, D. N., "Introduction to System of Systems Engineering Principles," , 2024.

[2] International Council on Systems Engineering, "System and SE Definitions," , 2024. URL https://www.incose.org/about-systems-engineering/system-and-se-definitions.

[3] Baker, A., "Model-Based Systems Engineering Introduction," , 2024.

[4] Zhang, J. A., "Leveraging SysML V2 for Integration of MBSE and Multidisciplinary System Development," *ARC Journal*, 2023. URL https://arc.aiaa.org/doi/abs/10.2514/6.2023-1895.

[5] MBSE4U, "SysML v2 Release: What's inside?" , 2024. URL https://mbse4u.com/2020/12/21/sysml-v2-release-whats-inside/.

[6] v2 API-Services, S., "SysML v2 API Services," , 2024. URL https://github.com/Systems-Modeling/SysML-v2-API-Services/.

[7] Microsoft, "Microsoft Learn OpenJDK 11," , 2024. URL https://learn.microsoft.com/en-us/java/openjdk/download#openjdk-11.

[8] OMG-SysML®-Developers, "OMG SysML® Version 2 Ecore based Meta Model Documentation," , 2024. URL https://modeldocs.sysml2.net/.

[9] Systems-Modeling, "SysML v2 API Cookbook," , 2024. URL https://github.com/Systems-Modeling/SysML-v2-API-Cookbook.

[10] Treelib-Developers, "Treelib Documentation," , 2024. URL https://treelib.readthedocs.io/en/latest/.

[11] PyGraphviz-Developers, "PyGraphviz Documentation," , 2024. URL https://pygraphviz.github.io/.

[12] Streamlit-Developers, "Streamlit Documentation," , 2024. URL https://streamlit.io/.