Advancing Model-based Engineering through Improved Integration of Domain-Specific Simulation and Analysis using SysML-based Models for Unmanned Aerial Vehicles

Daniel R. Herber*
Colorado State University, Fort Collins, CO 80523

Dominic Dierker†
PC Krause & Associates, Indianapolis, IN 46268

Soumya S. Patnaik‡
Air Force Research Laboratory, Dayton, OH 45433

In this paper, we describe an approach for integrating a variant-based Simulink model to simulate an unmanned aerial vehicle (UAV) with a SysML-based executable model in Cameo Systems Modeler. A variant-based model is one where different structural pieces (here termed subsystems) have various options available to simulate other behaviors and is a common step of the design process. For example, in the early-state design of an aircraft, we may have multiple different thermal management system (TMS) variants to consider. The overall approach is described in a tool-agnostic form, laying out specific requirements needed from the SysML modeling tool and external analysis tools to support the proposed integration methodology. While the simulation-focused benefits are valuable, we also outline several management and additional activities enhanced through the SysML-based modeling approach helping demonstrate how a principally simulation-based engineering group might consider the digital engineering transition and balance the benefits of SysML-based modeling approaches and existing domain-specific models.

I. Introduction

With an increasing reliance on unmanned aircraft vehicles (UAVs) for a large variety of military missions, the capability to design them for customized requirements is highly desirable. UAVs with a broad range of mission capabilities require diverse systems in terms of development cost and size. A wide range of mission capabilities also requires assessing a large variety of power and thermal technologies to support appropriate design decisions for technology selection. An agile and speedy design process is, therefore, integral to a robust modeling, simulation, analysis, and design capability as it allows for efficient identification of new technologies while simultaneously providing insight into improvement and modifications of existing technologies in a rapid and high confidence approach. Architecture-centric (and model-centric) approaches can be considered a potential solution to address this challenge.

Architecture-centric practices are gaining widespread acceptance in systems engineering (SE), but also in the engineering and management disciplines more broadly. This process involves capturing the structure, behavior, and rules and their relationships to create an abstract representation of a system, often termed a model of the system. A system model does not inherently provide value to the SE effort; rather, benefits are realized through its support of various SE activities, from understanding needs to solution test and deployment. To better define how to both synergistically create a system model and support the system development process, various model-based systems engineering (MBSE) practices are being explored [1–3]. Fortunately, there is codified and disciplined support for MBSE in the form of methodologies, formal modeling languages, and tools [1]. Overall, there is increasing formal and informal evidence that MBSE can improve efficiency, rigor, and quality during system development [3–5].

The Systems Modeling Language (SysML) is a widely-supported graphical modeling language that prescribes a set of object-oriented elements and allowable relationships between the different elements supporting the creation of an

*Assistant Professor, Systems Engineering, 200 W. Lake Street, 6029 Campus Delivery, Fort Collins, CO 80523, AIAA Member.
†Lead Engineer, 4291 W 96th Street, Indianapolis, IN 46268, AIAA Non-Member.
‡Principal Aerospace Engineer, Aerospace Systems Directorate, 1950 5th Street, Area B, Wright Patterson AFB, AIAA Senior member.

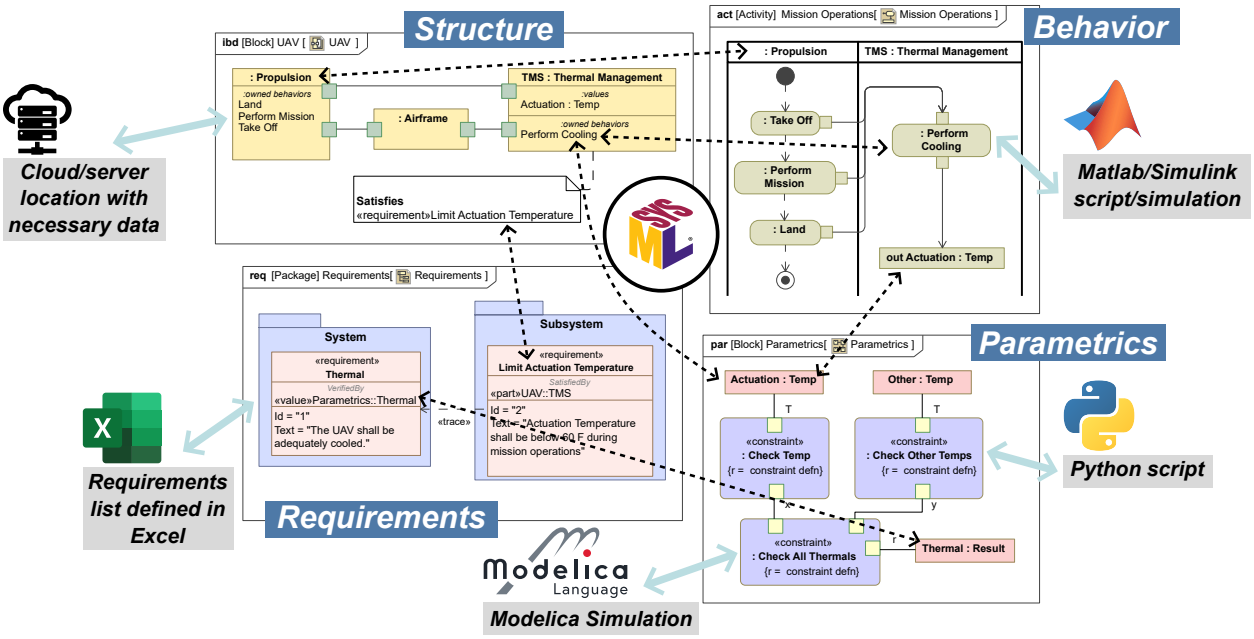


Fig. 1 SysML four pillars (requirements, structure, behavior, and parametrics) with additional relationships to external tools, environments, and data sources.

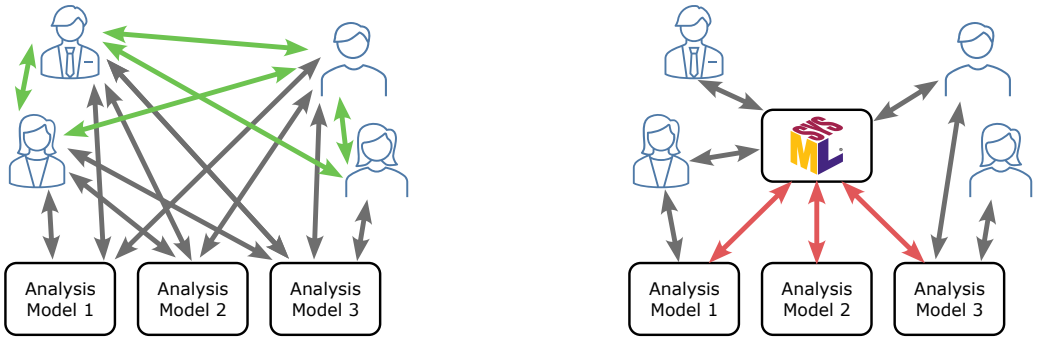
integrated system model that is interacted with through diagrams and other mechanisms [6, 7]. Software tools are the key enablers of this form of system modeling and MBSE. One popular tool (and the one used in this work) is Cameo Systems Modeler, which implements SysML along with other features to support MBSE activities [8]. Many of these tools are for building connections between the four pillars of SysML (requirements, structure, behavior, and parametrics) but also with external tools, environments, and data sources. This is conceptually visualized in Fig. 1. The advantages enabled by these additional relationships are the focus of this paper, particularly when computationally-intensive physics-based simulations are required.

There has been growing interest in adopting SysML and other similar MBSE tools for simulation-based engineering activities. Zhang et al. describe an approach for integrating domain-specific simulation models with SysML for wind turbines [9]. Ciampa et al. describe the AGILE project, a multidisciplinary design analysis optimization-focused approach for system development leveraging SysML techniques for formalizing and modeling the system and has been applied to the development of aircraft products [10, 11]. McKean et al. describe a similar approach utilizing a SysML-based tool to integrate various quantitative models (primarily in Matlab/Simulink) for performance estimates for computationally-intensive algorithms on an sCPU physical architecture [12]. There are also several efforts comparing MBSE and other tools that might be considered alternatives for the ones utilized in this work [13, 14]. However, these do not focus on the challenges regarding variant-based design (a critical system architecture decision). For example, you might have several discrete variants for a particular subsystem that you can choose between. Furthermore, a centralized SysML-based model enables additional management and communication mechanisms, even in a simulation-driven environment; such features are often discussed primarily in a systems engineering context.

The remainder of this paper is organized as follows. Section II presents the modeling approach for integrating SysML-based models with external analysis models that use a variant-based approach for defining different configurations. Section III discusses management and other activities through the SysML-based model. Section IV presents the results. Finally, Sec. V provides some conclusions and a future work discussion.

II. Integrating Analysis Models with SysML-based Tools

In this section, we will discuss the overarching framework for integrating an existing model-based engineering (MBE) program with SysML-based models for enhancing several aspects of the system development process. Figure 2 briefly highlights the fundamental distinction between a more traditional MBE approach and one where a SysML-based model serves as a central information source and coordination mechanism. In Fig. 2a, MBE activities are often coordinated



(a) Traditional. **(b) SysML-based.**

Fig. 2 Comparison between the relationships between models in traditional model-based engineering programs and one with the proposed SysML-based model as the single source of truth and integrator.

through human-centric practices where the complexity of these interactions can grow quickly as the number of distinct models and engineers responsible for them grows. On the other hand, in Fig. 2b, we place a SysML-based model at the center of the MBE activities to serve as the single source of truth (SSOT). Now, rather than a network of ad hoc formal and informal connections between the different engineering activities, we centralize *specific* pieces and formalize connections to the already existing models that are often developed in (and should remain in) domain-specific tools such as Matlab/Simulink, Modelica, etc.

From this discussion, there are several key components that define the composition of the software architecture:

- *SysML modeling tool* that supports the desired SysML version and other modeling capabilities [6]. Cameo Systems Modeler (CSM) v19.0 SP4 [8] supporting SysML v1.5 is used in this work.
- *Executable SysML model capabilities* that supports the desired interfacing and execution with the desired domain analysis tool(s). Cameo Simulation Toolkit (CST) v19.0 SP4 [15] is used in this work as it supports integration with Matlab.
- *External analysis tool(s)* for simulations that are desired to be integrated into the SysML model. Compatibility between the executable SysML model capabilities and external tools is required. Matlab/Simulink R2020b [16] is used in this work as it is the supported version for the existing aircraft subsystem simulation models.

Another critical aspect considered in this work is the common situation where multiple different modeling options are available for a given piece of a system. For example, in an aircraft, there is usually an engine. However, there are often multiple variations of engines that might be considered, especially during early-stage design. Seeing as the SysML-based model's key role is as an SSOT and integrator, we will outline an approach that supports the rapid assessment of different configuration options from the SysML-based model.

A. Variant-based Integrated (Simulink) Model

We begin with the definition of a variant-based integrated Simulink model that embodies the key simulation artifact that is to be interfaced with. To utilize this approach, one must first define the generalized top-level subsystems that comprise the system-of-interest. In this work on aircraft design, the four subsystems are 1) *Aerodynamics*, 2) *Engine*, 3) *Thermal Management System* (TMS), 4) and *Electrical Power System* (EPS). Specific information on the models used in each of these subsystems can be found at the end of this section. The variant-based approach considered here is an instance of the combining pattern from Ref. [17]. As an example, consider the following different sets of variants for the four different subsystems:

$$\{\text{Aero}\} \times \{\text{Engine-A, Engine-B}\} \times \{\text{TMS-A, TMS-B, TMS-C}\} \times \{\text{EPS-A, EPS-B}\} \tag{1}$$

which has 12 unique complete variant architectures using the Cartesian product with two specific architectures being {Aero, Engine-A, TMS-A, EPS-A} and {Aero, Engine-A, TMS-C, EPS-B}. Currently, all combinations are allowed, although network structure constraints could be defined as needed [18].

These subsystems do not operate in isolation but rather through a specific set of physical and information connections between the different subsystem features. To leverage variants, we must create a *variant architecture* for the system-of-interest, which defines, in a general enough way, the interfaces between the different subsystems. This description can

be visualized with an N^2 diagram and formally captured in the SysML model with interface blocks (see TMS Simulink Interface in Fig. 5 for one example). Additionally, there might be additional pieces that touch one or more subsystems, such as the mission profile for the aircraft. These should be defined as well and included in the completed description of a system configuration variant.

There are several important comments on this approach. First, the external analysis tool needs to support the programmatic selection of different variants, and the SysML-based model needs to be able to change or pass information that can change the variant selections. Here, the external analysis tool Simulink supports variant systems [19], and we change the variant selections through the SysML-based model by passing values to a Matlab script that performs the programmatic change. Standardized bus data structures are designed to pass inputs and outputs between the selected subsystems of interest within framework based on the predefined interfaces. Initialization scripts and a fixed directory structure allow users to easily swap between subsystem architecture selections and populate model boundary and initial conditions. Each subsystem has a unique folder where custom initialization scripts and data files can be stored. Top-level initialization scripts navigate to the selected subsystem folders to populate the model workspace and add the folder locations to the active Matlab path if required.

Second, since the variant-based Simulink model and scripts making changes are all within the Matlab/Simulink environment, an engineer primarily familiar with these tools can still utilize the integrated model with SysML capabilities, if necessary. Third, with the variant architecture defined, it is easy to communicate and integrate how a new variant can be incorporated, supporting more modular and rapid model-based development.

Another aspect to consider is how the results of the simulations will be returned to the SysML-based model. If the software architecture supports it, we could directly utilize the workspace for the external tool to get and manipulate the results from the SysML-based model. However, this is only temporary as the full simulation results would be lost unless saved to a particular location in the model or an appropriate data file. It might seem natural to save the results to the SysML-based model, but there are some important challenges with this approach. Since SysML treats more things as model elements, there is significant overhead with saving thousands of data points from a detailed simulation (e.g., a unique model identifier for each number). Furthermore, the data saved in the model is usually stored as characters rather than their native types (e.g., floating-point numbers). Therefore, in this approach, we primarily utilize data files (i.e., .mat files) created when a simulation completes and include all the relevant trajectories and variables used to define the system configuration. Also, having the data files available supports more efficient and modular storage.

Before describing the specific subsystem models, we first want to discuss an alternative approach that was prototyped before the presented singular integrated Simulink model. This alternative approach was based on the Functional Mock-up Interface (FMI), which is a free and open standard that "defines a container and an interface to exchange dynamic models" [20, 21]. The creation of Functional Mock-up Units (FMUs) was a natural fit for the different subsystem variants. For example, each variant in Eq. (1) is encapsulated as an FMU with the correct standardized interface. Then, in CSM, we could readily select between these FMUs and simulate [22] a combined architecture. While this generally functioned correctly, there were some key issues. First, during co-simulation, there was severe communication overhead between the different steps, causing long delays. Second, several control loops were split across the subsystem boundaries, and because CSM uses a fixed-step solver, tiny time steps were needed to ensure stability, producing much slower and less accurate results than the variable-step techniques available in Simulink. Therefore, this approach was abandoned in favor of the current CSM-Simulink interactions. This functionality could still be useful in the future if some of these issues are addressed and co-simulation between different groups or other model owners is required.

1. Aerodynamics

The aerodynamic subsystem is modeled using a reduced order vehicle model, specifically the two-and-a-half degree of freedom (2.5DOF) model developed by AFRL/RQQI and PCKA [23]. The 2.5DOF model is a point mass model built in Simulink that captures enough relevant aircraft physics to provide accurate range and endurance estimates in a fraction of the time that higher fidelity aerodynamic models would typically take. In addition, it is much easier to produce the weight and drag polar aerodynamics required by the 2.5DOF compared to the full set of aerodynamic force and moment data that would be needed for a standard six DOF model, which often requires time-consuming CFD runs to populate a full set of aerodynamic force and moment data.

2. Engine

The engine model used in the following trade studies is a Numerical Propulsion System Simulation (NPSS) table lookup model. The NPSS model includes sensitivities to bleed and shaft extraction to account for losses when extractions

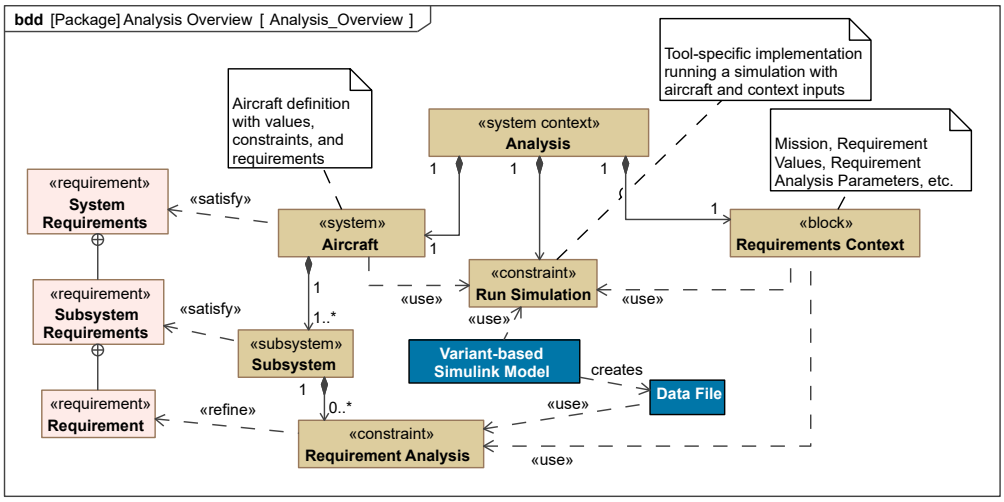


Fig. 3 Approach overview in the SysML-based model with the white text elements implemented in the external analysis tool (Simulink).

are required by the TMS or EPS subsystems. An NPSS five-column lookup table model uses the thrust commanded, the current Mach and altitude, and the required bleed and shaft extractions to calculate thrust, fuel burn, bleed flow properties, and the corresponding shaft speeds.

3. Thermal Management System

The TMS subsystem models used in the integrated studies are simulated using the ATTMOSphere library developed by AFRL/RQQM and PCKA, which is a high-fidelity, dynamic modeling toolbox for Simulink focused on aircraft power and thermal systems [24–28]. The initial design and sizing were performed using the UniSyS tool set, also developed by PCKA. Both tools feature a GUI that allows users to drag and drop components from an extensive library into a system model. With the architecture, boundary conditions, assumptions, and measurable system requirements defined, UniSyS is used to size specific TMS systems automatically. For the purposes of this study, UniSyS was used to size the system components and obtain mass, volume, flow, and power requirements for each TMS architecture at the selected design point. The UniSyS tool facilitated off-design analysis by providing a means to generate ATTMOSphere models of the sized systems automatically. Using this feature, all component sizing data from the UniSyS model is copied into the ATTMOSphere model and used to examine system-level impacts over a full dynamic mission.

The current work explores the integration differences between the preliminary designs of four traditional TMSs. In this work, a simple ram air cooled TMS, a vapor cycle system (VCS), an air cycle system (ACS), and a traditional air cycle system using ram air as the working fluid (Ram ACS) are explored. Variations of the baseline architectures include updated component sizing for different thermal loads and other minor architecture variations that are not detailed in this paper but make up a larger subset of architectures to be evaluated.

4. Electrical Power System

The EPS subsystem uses lower fidelity average value models built using an AFRL/RQQI Electrical Library and ASMG components. ASMG is an electrical component Simulink library developed by PCKA and based on [29]. Alternatively, to complete studies more rapidly if changes to the EPS are required, in-house tools are used to predict the size, weight, and power estimates for various EPS architectures.

B. SysML-based Model

We now briefly describe the key features of the SysML-based model for integrating with the analysis model from the previous section. A generalized structural and requirements overview is shown in Fig. 3. There are three pieces for a single analysis: 1) aircraft definition, 2) requirements context, and 3) simulation constraint (Run Simulation in the diagram).

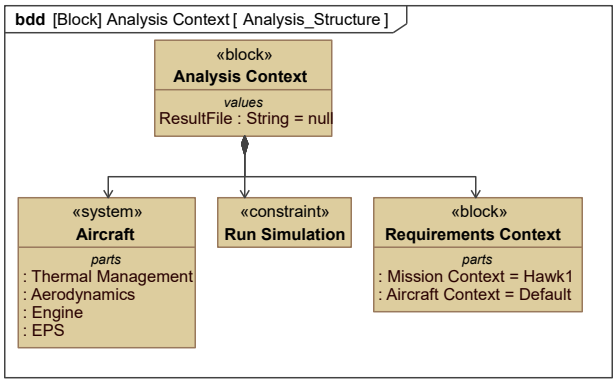


Fig. 4 Analysis structure.

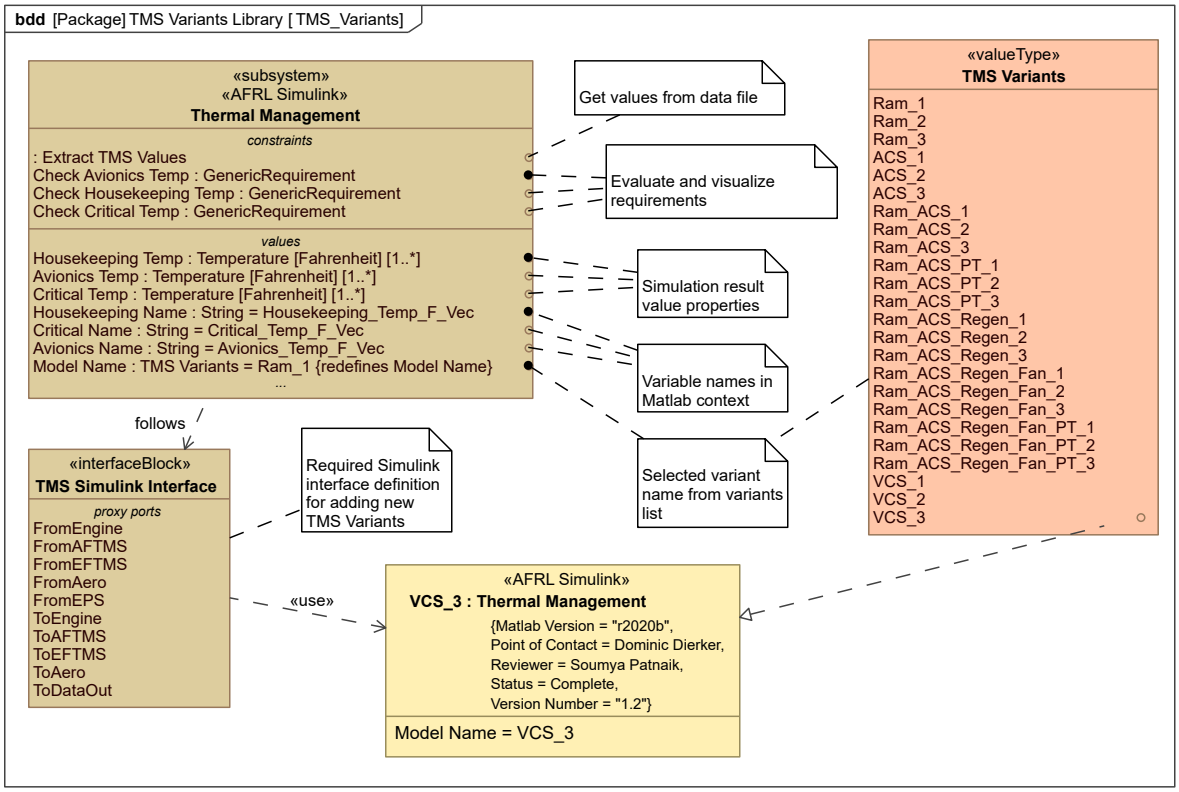


Fig. 5 Thermal management subsystem (TMS) definition with its variants.

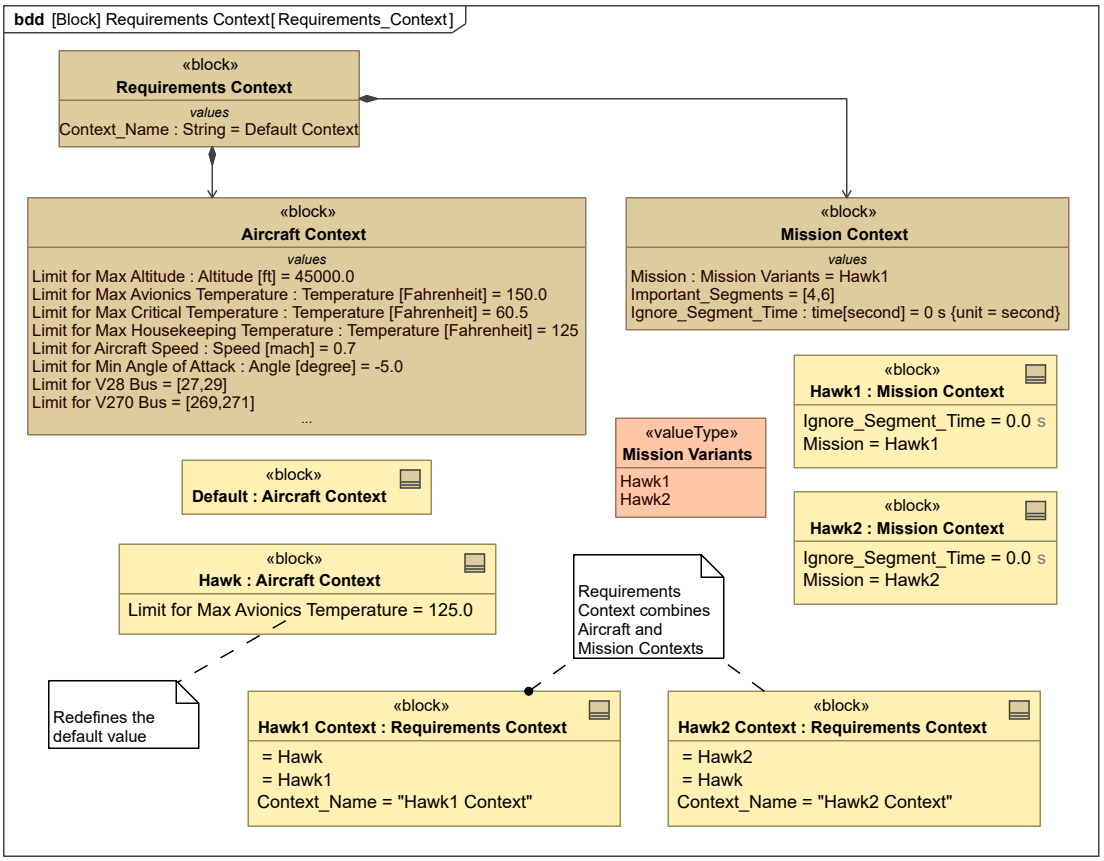


Fig. 6 Definition and usage of the Requirements Context block including both Aircraft and Mission Context blocks.

We will first describe the system definition (again, an aircraft in this work but can be replaced with a different system-of-interest). The system will have multiple subsystems, already described above in the context of the variant-based analysis model and visualized in the Aircraft block in Fig. 4. Each of the subsystems will have various requirement analysis constraint blocks that will refine a particular requirement, normally through the execution of some mathematical statement. Subsystems will also typically have other value properties, including the value that denotes which variant should be selected. In our implementation, we utilize Model Name. One subsystem is shown in Fig. 5 for the TMS. Note the various constraints for evaluating requirements, values for different temperatures, TMS Simulink Interface describing how the TMS is integrated into the variant architecture, and enumeration of TMS variant names and each of these is an available Simulink model like the options in Eq. (1).

Now, Fig. 3 indicates that there should be system, subsystem, and specific requirements with appropriate SysML relationships to the corresponding structural elements. This is a fairly straightforward requirements development, but we emphasize a certain flexibility in their definition as the specific limits and other factors will depend on the Requirements Context.

A Requirements Context block was defined to address the needs of rapidly changing requirement scenarios specifically. It is given a name and has two parts of Aircraft Context and Mission Context. Aircraft Context, in this implementation, includes the default limits for the various requirements (e.g., avionics temperature maximum value). This is visualized in Fig. 6 where Hawk : Aircraft Context redefines the limit for the maximum avionics temperature but utilizes the defaults for the remaining values. The Mission Context defines what mission should be considered (and therefore simulated) as well as some additional features to tailor how the mission is assessed. We can now utilize block instances to define different reusable requirement scenarios.

The final piece of Fig. 3 is Run Simulation, which should be an executable part of the model that takes the architecture variant description as an input and returns something when the simulation has been completed with the external analysis tool. For our approach, this is visualized with the parametric diagram in Fig. 7. The : Run Simulation constraint is linked

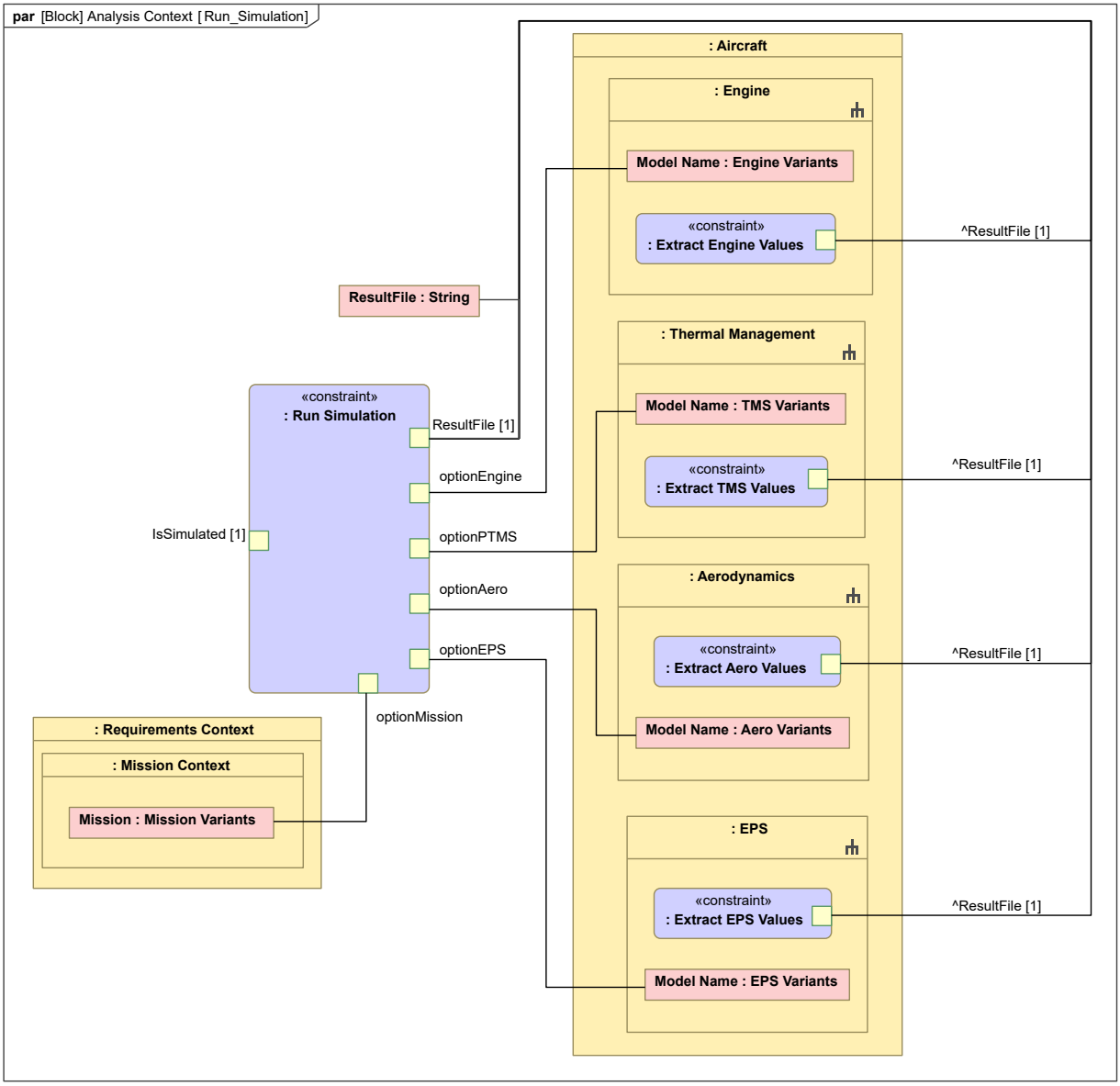
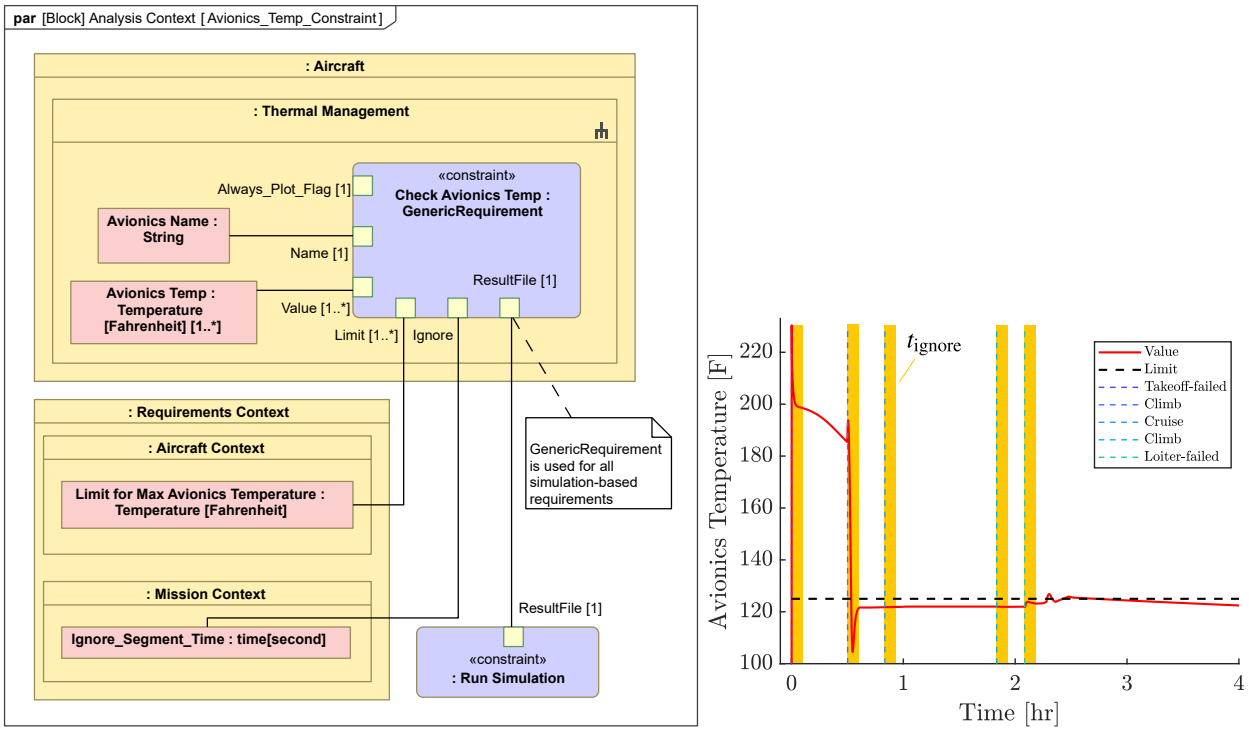


Fig. 7 Run Simulation executable constraint with links to the Requirements Context and Aircraft.



(a) Generic requirement «constraint» parametrics. **(b) Segment-based requirement visualization.**

Fig. 8 Illustration of the flexible requirements assessment approach for the avionics temperature signal.

to the Model Name for each of the subsystems (which are part properties of the aircraft). The mission option is a value property from Mission Context, as previously described. When the simulation completes, both ResultFile (string with the name of the .mat data created) and IsSimulated (Boolean value where true indicates the simulation has completed). These two values are also used to trigger the loading of the results into CSM and evaluation of the requirements. This constraint points to a Matlab script that does most of the heavy lifting by interpreting the aforementioned inputs, running a `sim` command on the integrated Simulink model, and saving the data file.

Now with the definition of the analysis approach in SysML defined, we can discuss how to execute the SysML-based model to obtain simulation results. There are currently three different modes for executing the analysis context. The first mode is for a single configuration where you select a value for each variant subsystem and Requirements Context. A user then runs the single simulation and visualizes the results within Cameo. The second and third modes are forms of trade studies. The second mode is a manual definition of multiple configurations that should be run, normally created and executed through an instance table. The final mode automatically generations different configurations by combining different variant sets (i.e., Cartesian product of all available options like the example in Eq. (1)).

C. Understanding Early-Stage Requirements through Simulations

Several goals were sought when developing an effective framework in the context of early-stage, potentially rapidly changing requirements.

First, as we will be dealing with time trajectories, there needs to be a way translate the time trajectory to a useful requirement Boolean assessment (i.e., pass/fail). The most straightforward way to do this is to take the maximum (and minimum) value a given signal reaches and compare that to a prescribed bound, which might represent the failure limits for some component or capability. Overall, the goal is to capture and alert an engineer if a particular combination of UAV subsystems, mission conditions, and requirements context is not passing a given test. However, at early stages of the design process, such limits might not be well defined, so flexibility is built in by defining a generic requirement assessment «`constraint`» that can be reused throughout the model (and fully described after the remaining goals are discussed).

Next, the simulations of the UAVs are often defined through discrete segments with specified boundary conditions

such as altitude and speed. A common categorization for these segments is ground idle, takeoff, climb, cruise, decent, loiter, and final approach. As a consequence of the transition between the boundary conditions, there is some transient behaviors immediately after the start of a new segment (e.g., a temperature spikes for a short period of time). Such a spike is often acceptable for the purposes of these simulations as long as the monitored signal is within the prescribed limits "quickly enough". For designer flexibility, this amount of time is parameterized with t_{ignore} in Fig. 8b. Additional requirements could be added to alert the engineer that these spikes are larger than desired rather than simply ignored. This framework also then leads to the potential to have segment-specific requirements rather than mission-only as is currently demonstrated.

Finally, we wanted to have effective visualization of the results beyond a table summarizing the pass/fail status, such as the one in Fig. 13. While SysML-based tools like CSM support visualizations with elements such as a time series chart, these approaches were found to be relatively expensive due to the amount of data that needed to be passed into the SysML tool as well as the lack of flexibility/customization in manipulating/presenting the data. Instead, with the generic requirement «`constraint`» implemented as a Matlab function, we can 1) customize the visualization of the given requirement (e.g., draw the segment boundaries and limit lines), 2) embed logic to check which segments the requirement failed on, and 3) reuse the function within the Matlab/Simulink environment as needed. The six inputs in Fig. 8a are then the trajectory data (`Value`), the upper and lower limits (`Limit`), t_{ignore} (`Ignore`), variable name in the result data file (`Name`), .mat file with the generated results (`ResultFile`), and a Boolean flag to plot the result even if the requirement does not fail (`Always_Plot_Flag`).

III. Management and Other Activities through the SysML-based Model

As stated in Sec. I, there are a variety of realizable advantages when deploying SysML-based models as the pillar for system development activities. Some of these advantages are described in Refs. [1, 7]. In this section, we will focus on some of the specific existing and new use cases realized in this work that better support the integration of analysis/ simulation/design-focused teams.

A. Standardization of Organizational Practices

Through the use of object-oriented principles and model-centric containers, standardization of many different kinds of organizational practices can be achieved. There are many examples which include:

- Interface definitions (see Fig. 5) required for new analysis model instances, including the ports, units, and interpretation, which are now well-defined and easily communicated.
- Definition of unit tests for individual analysis models and their expected results can be included to help ensure that required libraries/licenses are up-to-date and working.
- Customized «`Managed`» stereotype in Fig. 9 to help manage different model elements and diagrams (and indirectly the external things they might refer to). This currently includes the point of contact for the element (which should be an «`AFRL Actor`» described below), reviewer, status from an enumerated list of options, version number, and review dates.
- Customized requirement «`AFRL Requirement`», model «`AFRL Simulink`», and actor «`AFRL Actor`» stereotypes with tags to appropriately capture needed information. This is visualized in Fig. 9 for both the definition and example of the tags included for organization specific requirements and simulation models. Note that there is a generalization relationship between «`AFRL Requirement`»/«`AFRL Simulink`» and «`Managed`» so each of those stereotypes will also have the tags for management. This idea could be applied to other element types as needed.

B. Documentation

As digital engineering and model-centric practices increase in their adoption, there is the opportunity and challenge of maintaining current and consistent practices throughout an organization. In many cases, this is addressed through documentation.

There are two distinct features in SysML-based models that can be leveraged for more effective documentation in a model-driven environment. First, you can provide customized documentation *directly* with their definition (e.g., with models, interfaces, requirements, etc.) rather than in a secondary location. Properties such as rationale, assumptions, maintainer, version, to-do items, etc. can all be included directly with comments or customized tags. Second, the overall documentation effort can be minimized because effort can be spent describing each element well and providing the correct relationships to other elements that contain their own appropriate documentation. The common question of

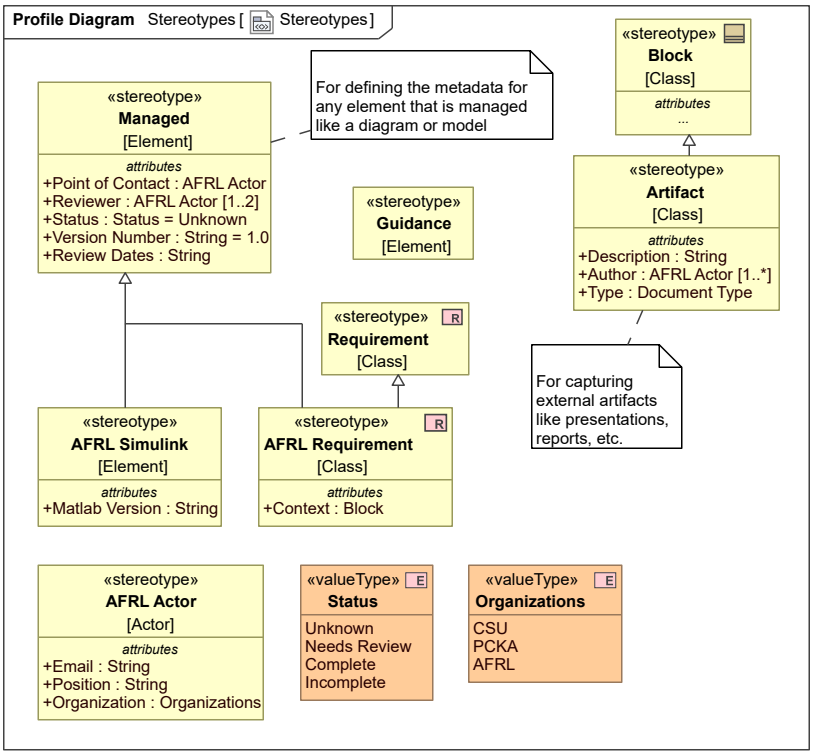


Fig. 9 Customized stereotypes for improved standardization and documentation.

when should we stop documenting is a bit clearer when this boundary is well-defined and the knowledge that the other interacting elements contain the needed information. An example of this concept is shown in Fig. 10, which captures the different steps in the workflow for running a desired configuration with comments directly hyperlinking the relevant model content to the steps.

Appropriately utilized, a centralized SysML-based model can also serve as the cornerstone for a more unified understanding of the system and its related development activities. For example, a glossary was developed for the system considered in Sec. II, and abbreviations such as "Hawk" and "TMS" are often utilized. Many SysML-based modeling tools support glossaries which allow an organization or team to build up a tailored list of terms that help reduce the technical gap between novice and advanced users. Furthermore, when these terms are detected in the model, a visual indicator is provided, letting users know they can click on the term to see its definition.

C. Stakeholder Dialog and Artifacts

One of the more commonly cited advantages of SysML-based modeling is the ability to tailor the model and its presentation to a variety of different stakeholder viewpoints. Some stakeholders are more concerned with the details of a specific analysis/modeling domain (e.g., domain specialists), while others are more concerned with the interfaces between models (e.g., system engineers or managers). Other key stakeholders include management and external collaborators/contractors who often expect a very different presentation of the underlying system and results. Often, significant resources are expended to support different types of discussions, reviews, etc. With a central model, specific diagrams can be readily created to support the right kind of dialog between stakeholders. Details can be readily hidden or shown in views that do not modify the underlying system definition. Stereotypes can be leveraged to tag model content appropriate for certain stakeholders, with a common example being different views for mechanical, electrical, control, and other domains. Additionally, novel dynamic interactions are made possible when the model itself is being reviewed rather than artifacts generated from it. When questions (naturally) arise, it is straightforward to navigate through the model to find the answers or exercise the model to create specialized views on the fly (e.g., dependency matrix that shows what simulation outputs are linked to requirements).

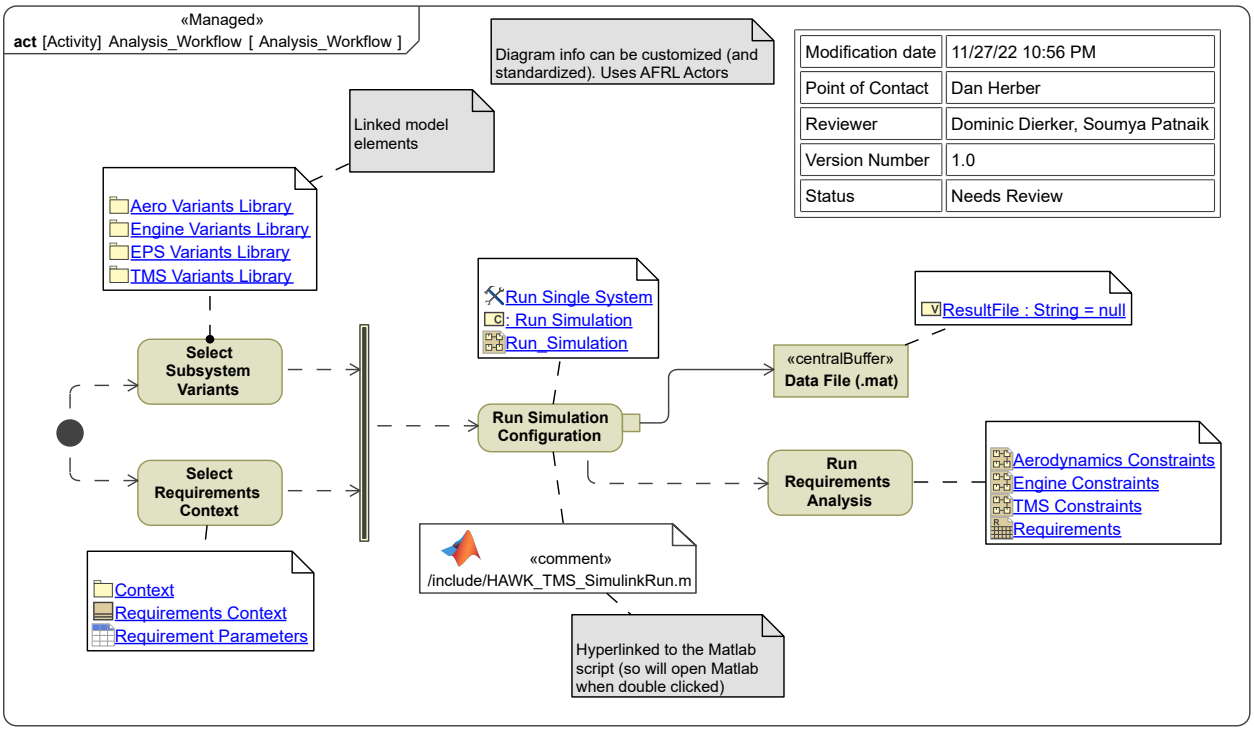
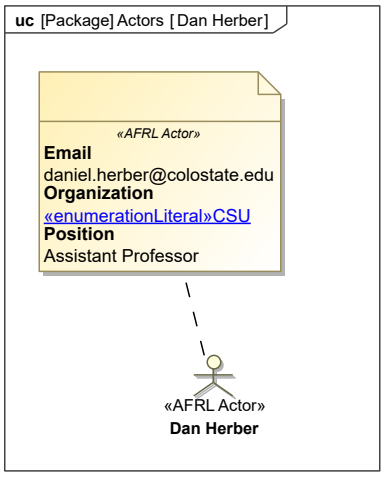
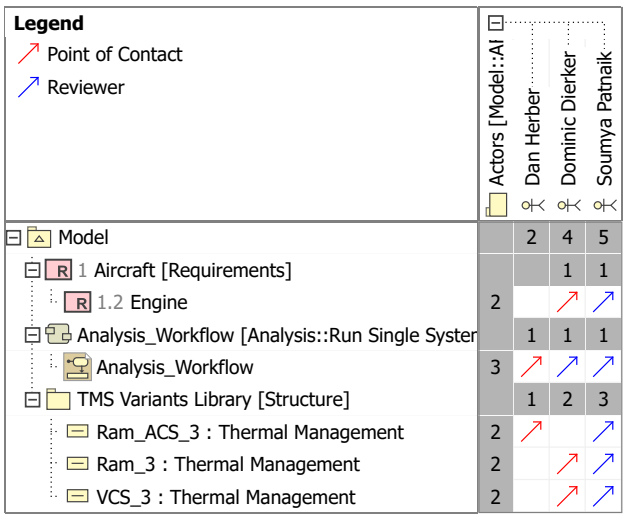


Fig. 10 An activity diagram showing the workflow for running a simulation with hyperlinked model and external elements for additional context/documentation.



(a) Example of an «AFRL Actor».



(b) Autogenerated summary matrix of Point of Contact and Reviewer relationships.

Fig. 11 Definition and usage of «AFRL Actor» actors for managing various model elements such as requirements, workflows, and model variants.

As an example, consider Fig. 11b, which summarizes the «Managed» elements in the model which can include requirements, workflows, and model variants and whomever (as defined by an «AFRL Actor» actor) is assigned as the Point of Contact and Reviewer. This view into the SysML model can be readily customized to show only a subset of the different element types (e.g., only the instances in the TMS Variants Library package). This view could also be used to assign or modify the Point of Contact and Reviewer information during a group meeting and then be readily available as it is captured in the same model as other engineering activities.

Furthermore, many SysML tools support automated artifact generation, which is often still relied upon. For example, if you need an up-to-date summary of the current domain models available and the required interface, a single export command in the model can produce a document or presentation that contains this information. Detailed reports based on the content can be formatted and then automatically generated when needed. Artifact generation does not necessarily need to be static images. Some tools support the exporting of a self-contained HTML-based view of the model that can be shared and read by anyone with an internet browser [30]. Overall, there are several novel mechanisms enabled through SysML-based tools for communication with *everyone* that can reduce the resources spent creating these artifacts as well as minimizing translation mistakes.

IV. Results

In this section, we will demonstrate the technical feasibility of the proposed approach combining an integrated variant-based Simulink model with a SysML-based model for various engineering activities. A total of 16 different analysis context configurations were set in the SysML-based model and then simulated with Simulink in an automated manner. The 16 combinations are a product of the 4 different TMS subsystem variants (and 30+ are currently being integrated), 2 different EPS variants (an empty and a specific nonempty case), and 2 different flight missions.

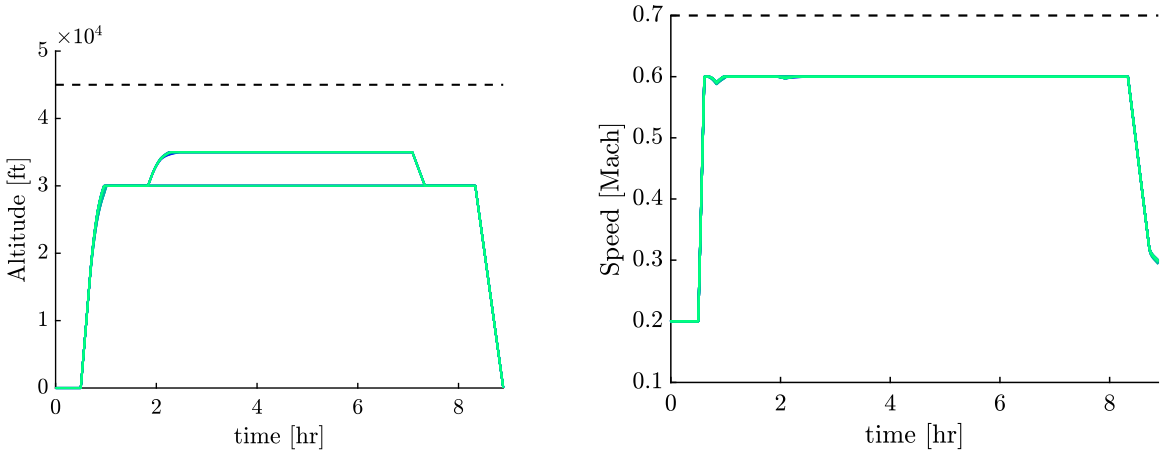
Figure 12 highlights the 16 solutions with several key trajectories visualized. Each of the different configurations evaluated defined in Fig. 13. Figures 12a and 12b show the altitude and speed of the aircraft, respectively. We note that there are two main missions, so we see only two general paths with some minor variations depending on the configuration. More interestingly, Figures 12c and 12d highlight the avionics and critical load temperatures, respectively. Here we have much more diversity in results, with each configuration producing distinct results. Note that while many configurations generally remain below the requirement limit, some exceed it both near segment changes and in a steady state. Finally, Figures 12e and 12f visualize the fuel reserves and voltage for one of the buses when the EPS is nonempty. Such visualizations, along with tables such as Fig. 13 which highlights the failure of specific requirements, will help engineers rapidly explore different combinations in the face of ever-changing needs and ever-developing models.

An important consideration when utilizing a non-native tool/language/environment for executing simulations is the overhead cost than naturally comes with this additional layer. Here we currently measure nearly zero overhead running a simulation since only a single Simulink `sim` command is executed for a selected combination of subsystems and mission. This statement is very different than a co-simulation approach that was initially pursued in this work utilizing the Functional Mock-up Interface (FMI), as mentioned in Sec. II.A. There is a small additional cost to import the data and evaluate the requirements within the selected tool (i.e., CSM). Ensuring a minimal hit to computational performance was an important driver as the engineering group under consideration has a wide variety of computationally-demanding models where significant slowdowns would have been deemed unacceptable.

As discussed in Sec. III, the goal here is not only to explore the technical feasibility of integrated configuration simulations but also the management and other activities through the SysML-based as the SSOT. Anecdotally, the SysML model has enabled better communication and a more rapid generation of artifacts for various reviews and presentations. Still, future work remains to explore the added value of the SysML-informed development process formally.

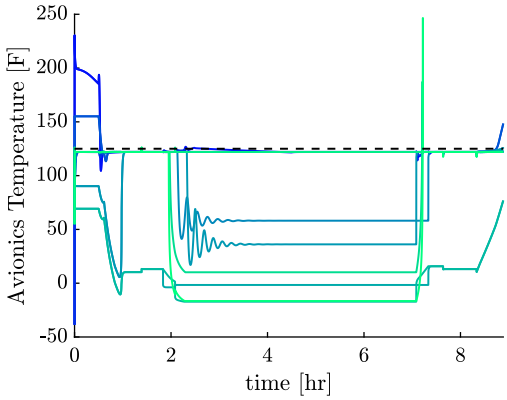
V. Conclusion and Future Work

In this work, we have explored how modern model-based engineering and model-based systems engineering activities will be better supported with both formal and informal integration with SysML-based modeling techniques and tools. The demonstration application using SysML in Cameo Systems Modeler linked with detailed physics-based Simulink models focused on UAVs. This approach supported the definition and simulation of 16 distinct configurations (with the potential of 100+ configurations soon with additional subsystem variants being added) along with support for many other typical management and engineering activities. While there will undoubtedly be some upfront work determining how a particular organization's practices should be enhanced through these mechanisms, the initial costs

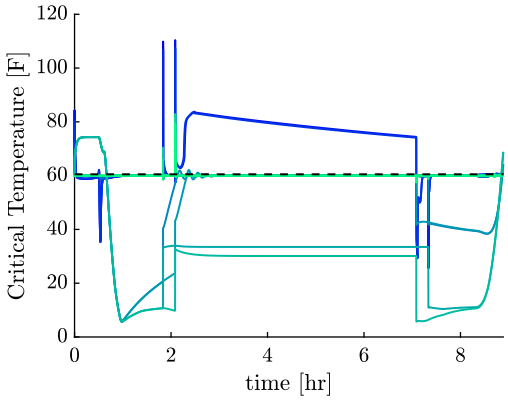


(a) Aircraft altitude.

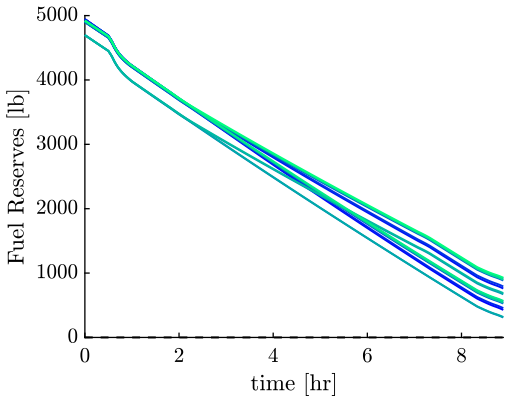
(b) Aircraft speed.



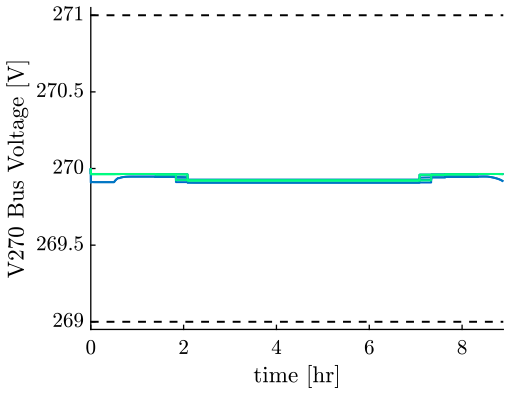
(c) Avionics temperature.



(d) Critical temperature.



(e) Fuel reserves.



(f) V270 Bus.

Fig. 12 Various trajectories for the 16 different configurations simulated with dash line indicating the considered requirements limit.

#	Name	:Requirements Context.:Mission Context : Mission Context	:Aircraft.:Aerodynamics.Model Name : Aero Variants	:Aircraft.:Engine.Model Name : Engine Variants	:Aircraft.:Thermal Management.Model Name : TMS Variants	:Aircraft.:EPS.Model Name : EPS Variants	ResultFile : String	:Aircraft.:Thermal Management.Check Avionics Temp : GenericRequirement	:Aircraft.:Thermal Management.Check Housekeeping Temp : GenericRequirement
1	Analysis 1	Hawk1 : Mission Context	Aero_1	Eng_1	ACS_3	Empty	Results-Architecture-Aero_1-ACS_3-Eng_1-Empty-Hawk1-1669608882	fail	pass
2	Analysis 2	Hawk2 : Mission Context	Aero_1	Eng_1	ACS_3	Empty	Results-Architecture-Aero_1-ACS_3-Eng_1-Empty-Hawk2-1669600025	fail	pass
3	Analysis 3	Hawk1 : Mission Context	Aero_1	Eng_1	ACS_3	Hawk_buck_dual	Results-Architecture-Aero_1-ACS_3-Eng_1-Hawk_buck_dual-Hawk1-1669608434	pass	fail
4	Analysis 4	Hawk2 : Mission Context	Aero_1	Eng_1	Ram_ACS_3	Hawk_buck_dual	Results-Architecture-Aero_1-Ram_ACS_3-Eng_1-Hawk_buck_dual-Hawk2-1669600779	pass	fail
5	Analysis 5	Hawk1 : Mission Context	Aero_1	Eng_1	Ram_ACS_3	Empty	Results-Architecture-Aero_1-Ram_ACS_3-Eng_1-Empty-Hawk1-1669607201	fail	pass
6	Analysis 6	Hawk2 : Mission Context	Aero_1	Eng_1	Ram_ACS_3	Empty	Results-Architecture-Aero_1-Ram_ACS_3-Eng_1-Empty-Hawk2-1669601102	fail	pass
7	Analysis 7	Hawk1 : Mission Context	Aero_1	Eng_1	Ram_ACS_3	Hawk_buck_dual	Results-Architecture-Aero_1-Ram_ACS_3-Eng_1-Hawk_buck_dual-Hawk1-1669606838	pass	fail
8	Analysis 8	Hawk2 : Mission Context	Aero_1	Eng_1	Ram_3	Hawk_buck_dual	Results-Architecture-Aero_1-Ram_3-Eng_1-Hawk_buck_dual-Hawk2-1669601394	pass	fail
9	Analysis 9	Hawk1 : Mission Context	Aero_1	Eng_1	Ram_3	Empty	Results-Architecture-Aero_1-Ram_3-Eng_1-Empty-Hawk1-1669605978	pass	pass
10	Analysis 10	Hawk2 : Mission Context	Aero_1	Eng_1	Ram_3	Empty	Results-Architecture-Aero_1-Ram_3-Eng_1-Empty-Hawk2-1669601574	pass	pass
11	Analysis 11	Hawk1 : Mission Context	Aero_1	Eng_1	Ram_3	Hawk_buck_dual	Results-Architecture-Aero_1-Ram_3-Eng_1-Hawk_buck_dual-Hawk1-1669605765	pass	fail
12	Analysis 12	Hawk2 : Mission Context	Aero_1	Eng_1	Ram_3	Hawk_buck_dual	Results-Architecture-Aero_1-Ram_3-Eng_1-Hawk_buck_dual-Hawk2-1669601870	pass	fail
13	Analysis 13	Hawk1 : Mission Context	Aero_1	Eng_1	VCS_3	Empty	Results-Architecture-Aero_1-VCS_3-Eng_1-Empty-Hawk1-1669605385	pass	pass
14	Analysis 14	Hawk2 : Mission Context	Aero_1	Eng_1	VCS_3	Empty	Results-Architecture-Aero_1-VCS_3-Eng_1-Empty-Hawk2-1669602270	fail	fail
15	Analysis 16	Hawk2 : Mission Context	Aero_1	Eng_1	VCS_3	Hawk_buck_dual	Results-Architecture-Aero_1-VCS_3-Eng_1-Hawk_buck_dual-Hawk2-1669603146	fail	fail
16	Analysis 15	Hawk1 : Mission Context	Aero_1	Eng_1	VCS_3	Hawk_buck_dual	Results-Architecture-Aero_1-VCS_3-Eng_1-Hawk_buck_dual-Hawk1-1669604729	fail	fail

Fig. 13 Results table for the 16 different configurations simulated focusing on configuration specification and thermal management system requirements.

should have substantial long-term benefits as systems that we are tasked with developing grow in complexity (as well as the engineering practices and tools that we use to make informed decisions). With improved standardization, reuse, and communication, the proposed SysML-based modeling approach can better align the needs of current simulation-based teams with well-accepted systems engineering principles to meet the needs of modern system development.

There are several key directions that would help move the suggested SysML-based framework toward the overarching goal of more effective digital engineering in model and simulation-heavy organizations. First, as with many digital approaches with models, software, and data, integrating the proposed approach with version control would be desirable. Due to their XML-based or proprietary file formats, many of the models considered here (both SysML and Simulink) often have tool-specific version control and diff tools. Therefore, future work should pursue a practical approach for realizing version control, including keeping different models/data up-to-date from within the different environments. This feature could include integrating physical test data seamlessly into system development and, thus, developing a digital twin artifact.

Furthermore, many model-based organizations develop their models intending to explore various possibilities, such as gains in a controller or sizing of a physical component. Such design tasks are often coupled between the architecture simulations and some systematic method for exploration (e.g., design optimization algorithms). Unfortunately, many SysML environments are lacking in this respect, but there are alternative tools that focus more directly on optimization and other related tasks. Tools such as ModelCenter MBSE [31] have several capabilities related to integrating SysML models and optimization, but a holistic digital engineering strategy should best balance the factors described in this paper.

Additionally, as the specific modeling approach used in this study was primarily within Mathworks tools, further investigations into first-party products like the Requirements Toolbox [32] and Simulink Test [33] might prove helpful even if they have some limitations compared to the SysML modeling approach. Maintaining the SSOT is still possible by utilizing standards such as the Requirements Interchange Format (ReqIF) [34] for tool-agnostic representations. Addressing some of the concerns regarding the FMU-based approach (see Sec. II.A) would also allow for the usage of multiple distinct simulation tools.

Funding Sources

The authors gratefully acknowledge funding and support from AFRL/RQQI.

Acknowledgments

Special thanks to E. K. Iskrenova-Ekiert and Brian Raczkowski for helpful discussions and support in model building.

References

[1] Borky, J. M., and Bradley, T. H., *Effective Model-Based Systems Engineering*, Springer, 2019. https://doi.org/10.1007/978-3-319-95669-5.

[2] Navas, J., Bonnet, S., Voirin, J.-L., and Journaux, G., "Models as enablers of agility in complex systems engineering," *INCOSE International Symposium*, Vol. 30, No. 1, 2020, pp. 339–355. https://doi.org/10.1002/j.2334-5837.2020.00726.x.

[3] Huldt, T., and Stenius, I., "State-of-practice survey of model-based systems engineering," *Syst. Eng.*, Vol. 22, No. 2, 2018, pp. 134–145. https://doi.org/10.1002/sys.21466.

[4] Carroll, E. R., and Malins, R. J., "Systematic literature review: how is model-based systems engineering justified?" Tech. Rep. SAND2016-2607, Sandia National Laboratories, Mar. 2016. https://doi.org/10.2172/1561164.

[5] Madni, A., and Purohit, S., "Economic analysis of model-based systems engineering," *Systems*, Vol. 7, No. 1, 2019. https://doi.org/10.3390/systems7010012.

[6] Object Management Group, "OMG System Modeling Language v1.5," , May 2017. URL https://www.omg.org/spec/SysML/1.5/.

[7] Friedenthal, S., Moore, A., and Steiner, R., *A Practical Guide to SysML*, 3rd ed., Elsevier, 2015. https://doi.org/10.1016/c2013-0-14457-1.

[8] No Magic, "19.0 LTR SP4 Version News," Online, Jun. 2020. URL https://docs.nomagic.com/display/CSM190SP4/19.0LTRSP4VersionNews.

[9] Zhang, Y., Hoepfner, G., Berroth, J., Pasch, G., and Jacobs, G., "Towards holistic system models including domain-specific simulation models based on SysML," *Systems*, Vol. 9, No. 4, 2021. https://doi.org/10.3390/systems9040076.

[10] Ciampa, P. D., and Nagel, B., "AGILE Paradigm: The next generation collaborative MDO for the development of aeronautical systems," *Progress in Aerospace Sciences*, Vol. 119, 2020, p. 100643. https://doi.org/10.1016/j.paerosci.2020.100643.

[11] Ciampa, P. D., Rocca, G. L., and Nagel, B., "A MBSE approach to MDAO systems for the development of complex products," *AIAA AVIATION Forum*, 2020. https://doi.org/10.2514/6.2020-3150.

[12] McKean, D., Moreland, J. D., and Doskey, S., "Use of model-based architecture attributes to construct a component-level trade space," *Systems Engineering*, Vol. 22, No. 2, 2019, pp. 172–187. https://doi.org/10.1002/sys.21478.

[13] Vaneman, W. K., Carlson, R. R., and Parker, G. W., "Model based systems engineering tool study report," Tech. rep., Naval Postgraduate School, 2021.

[14] Lu, J., "Research survey on model-based systems engineering tool-chain," Tech. rep., 2019.

[15] No Magic, "19.0 LTR SP4 Version News," Online, Jun. 2020. URL https://docs.nomagic.com/display/CST190SP4/19.0LTRSP4VersionNews.

[16] MathWorks, "R20120b at a glance," Online, 2020. URL https://www.mathworks.com/products/new_products/release2020b.html.

[17] Selva, D., Cameron, B., and Crawley, E., "Patterns in system architecture decisions," *Systems Engineering*, Vol. 19, No. 6, 2016, pp. 477–497. https://doi.org/10.1002/sys.21370.

[18] Herber, D. R., Allison, J. T., Buettner, R., Abolmoali, P., and Patnaik, S. S., "Architecture generation and performance evaluation of aircraft thermal management systems through graph-based techniques," *AIAA Scitech 2020 Forum*, 2020. https://doi.org/10.2514/6.2020-0159.

[19] MathWorks, "Variant systems," Online, 2022. URL https://www.mathworks.com/help/simulink/variant-systems.html.

[20] Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., and Viel, A., "Functional mockup interface 2.0: the standard for tool independent exchange of simulation models," *Linköping Electronic Conference Proceedings*, 2012. https://doi.org/10.3384/ecp12076173.

[21] Modelica Association, "Functional Mock-up Interface Specification: version 3.0, 2022-05-10," Online, May 2022. URL https://fmi-standard.org/docs/3.0/.

[22] No Magic, "FMI 2.0 co-simulation," Online, 2022. URL https://docs.nomagic.com/display/CST190/FMI+2.0+co-simulation.

[23] Shimmin, K., Russell, G., Reuter, R. A., and Iden, S., "Development and performance of a reduced order dynamic aircraft model," *SAE Technical Paper Series*, 2015. https://doi.org/10.4271/2015-01-2415.

[24] McCarthy, P. T., McCarthy, K., Hasan, M., Boyd, M., Chang, M., Walters, E., and Niedbalski, N., "A multi-domain component based modeling toolset for dynamic integrated power and thermal system modeling," *SAE Technical Paper Series*, 2019. https://doi.org/10.4271/2019-01-1385.

[25] McCarthy, K., Walters, E., Heltzel, A., Elangovan, R., Roe, G., Vannice, W., Schemm, C., Dalton, J., Iden, S., Lamm, P., Miller, C., and Susainathan, A., "Dynamic thermal management system modeling of a more electric aircraft," *SAE Technical Paper Series*, 2008. https://doi.org/10.4271/2008-01-2886.

[26] McCarthy, P., Niedbalski, N., McCarthy, K., Walters, E., Cory, J., and Patnaik, S., "A first principles based approach for dynamic modeling of turbomachinery," *SAE International Journal of Aerospace*, Vol. 9, No. 1, 2016, pp. 45–61. https://doi.org/10.4271/2016-01-1995.

[27] Kania, M., Koeln, J., Alleyne, A., McCarthy, K., Wu, N., and Patnaik, S., "A dynamic modeling toolbox for air vehicle vapor cycle systems," *SAE Technical Paper Series*, 2012. https://doi.org/10.4271/2012-01-2172.

[28] McCarthy, K., McCarthy, P., Wu, N., Alleyne, A., Koeln, J., Patnaik, S., Emo, S., and Cory, J., "Model accuracy of variable fidelity vapor cycle system simulations," *SAE Technical Paper Series*, 2014. https://doi.org/10.4271/2014-01-2140.

[29] Wasynczuk, O., and Sudhoff, S. D., "Automated state model generation algorithm for power circuits and systems," *IEEE Transactions on Power Systems*, Vol. 11, No. 4, 1996, pp. 1951–1956. https://doi.org/10.1109/59.544669.

[30] No Magic, "Generating web report," Online, 2022. URL https://docs.nomagic.com/display/SYSMLP190/Generatingwebreport.

[31] Ko, A., Keel, W., and Beale, J., "ModelCenter MBSE for OpenMBEE: MBSE analysis integration for distributed development," *AIAA AVIATION 2022 Forum*, 2022. https://doi.org/10.2514/6.2022-3235.

[32] MathWorks, "Requirements toolbox," Online, 2022. URL https://www.mathworks.com/products/requirements-toolbox.html.

[33] MathWorks, "Simulink test," Online, 2022. URL https://www.mathworks.com/products/simulink-test.html.

[34] Object Management Group, "Requirements interchange format," Online, Jul. 2016. URL https://www.omg.org/spec/ReqIF/, version 1.2.