Model-Based Integration

The Aerospace Corporation

Achyut Patel
Industrial and Systems Engineering
Virginia Tech
Blacksburg, VA, United States
amp10@vt.edu

Alex Harfouche
Industrial and Systems Engineering
Virginia Tech
Blacksburg, VA, United States
galex21@vt.edu

Derek Spratley
Industrial and Systems Engineering
Virginia Tech
Blacksburg, VA, United States
dsprat@vt.edu

I. Introduction

The Aerospace Corporation is a leading nonprofit organization providing technical expertise and advanced engineering solutions to government and civil space programs. Established in 1960 and headquartered in El Segundo, California, Aerospace operates as the only Federally Funded Research and Development Center (FFRDC) for the space sector.

Their objective is to ensure mission success in space exploration, satellite systems, and space technologies. These efforts often involve customers like the Department of Defense, U.S. Air Force, and U.S. Space Force. The company specializes in areas such as systems engineering, spacecraft design, space mission assurance, cybersecurity, and research.

One large component of the company is the systems engineering department which leverages advanced Model-Based Systems Engineering (MBSE) methodologies to drive innovation and efficiency in complex projects, such as this model-based integration project. These MBSE models are used to convey information and data to company clients regarding models and digital systems. By employing MBSE, the team enhances collaboration and streamlines the decision-making process throughout project life cycles. The ability to effectively communicate this information is of the utmost importance because it allows for efficient use of time and resources between Aerospace and their customers.

The current system involves Aerospace forwarding project or system files to the client where they are then required to interpret the information themselves. They use one of the leading MBSE model software packages called Cameo Systems Modeler, primarily employed in the aerospace and defense sectors. The licensing of this software is incredibly costly for both Aerospace and their clients who need to access models. Not only is it expensive, but complex and out-of-date interfaces make it difficult for new users to find the information they need. This is a problem because information is conveyed incorrectly and inefficiently which results in problems for both entities. These challenges prompted Aerospace to investigate possible solutions.

It was determined that a digital interface, in the form of an interactive dashboard, would be an effective approach. This dashboard interface would process model data from the selected files and display it through a series of dynamic graphical representations. The interactive nature of the interface will allow their customers to customize their view, selecting information that is relevant to their needs. By leveraging this tool, clients will be able to interpret information more efficiently. This reduces confusion and fosters greater collaboration and transparency between both parties.

II. Problem

A. Problem Statement

The Aerospace Corporation, a federally funded research and development organization in aerospace and defense, currently relies on proprietary software with complex interfaces for accessing Model-Based Systems Engineering (MBSE) models. This approach presents significant challenges for non-technical stakeholders, who often lack the training or access needed to engage with the software. As a result, their ability to understand and contribute to the models is limited, hindering collaboration and informed decision-making. These issues are leading to unmet expectations, lost business opportunities, and decreased sales.

B. Objectives

We identified two measurable objectives to ensure the project meets client requirements. The first objective is to eliminate software barriers. Our web-based application needs to be simple and user-friendly, making it accessible even for non-technical users. To measure this, we are using the System Usability Scale (SUS) questionnaire. Initial responses from Aerospace employees familiar with Cameo provide a baseline, and these users will complete the questionnaire again after the application is finalized to evaluate improvements.

The second objective is to achieve cost savings. The client estimates these savings to range from $100,000 to $500,000, primarily by reducing the need for multiple Cameo licenses, which cost $5,000–$10,000 each. This objective is going to be measured by working closely with the client and using a cost calculator they have developed.

C. Requirements

The deliverable for this project is the web-based application that interacts with MBSE models exported from Cameo.

The requirements are that this application/dashboard has an intuitive and user-friendly interface. This is important as the goal is to enable non-technical stakeholders to access and utilize the data contained in the models. Additionally, another requirement is that it comes at no extra cost, time, or inconvenience to the non-technical stakeholders. This is particularly important for decision-makers who lack the time or inclination to learn tools like Cameo. Finally, the dashboard and software developed should work seamlessly with models of all sizes and complexity.

D. Scope

The project scope focuses specifically on delivering a proof-of-concept system that demonstrates the feasibility of extracting key information from Cameo Systems Modeler and presenting it in a new, user-friendly, and free-to-use environment. The primary objective is to validate the technical approach and interface design to ensure that diagrams, requirements, and system data can be parsed, processed, and visualized effectively without relying on Cameo software.

The automatic refresh of data within the dashboard was considered out of scope, provided that a simple and effective method for manual data reloading could be identified. Our goal was to demonstrate that model data could be extracted from Cameo and rendered in a usable, interactive format. Therefore, developing a fully automated pipeline for detecting changes in model files and dynamically updating the dashboard was not pursued.

Additionally, security and access control measures for the dashboard were also outside the scope of this project. Since the final deployment and hosting environment will be managed by The Aerospace Corporation, all security protocols, user authentication mechanisms, and data protection will be addressed by their internal cybersecurity teams upon completion.

III. Approach

A. Methodology

For this project, we decided to use the Scrum methodology, a commonly used framework that emphasizes incremental progress and flexibility. Scrum was selected because it is an adaptive approach with benefits towards complex iterative design and development, like what is commonly required in software development.

When selecting a design methodology, we carefully evaluated various options to determine the best fit for our team. We decided to prioritize a method that allows extensive collaboration, which is critical for a small team, while also allowing for sequential progress. Because this is a software-based project, we felt that an iterative methodology would work best for us. When comparing alternatives, our choices included: Scrum, Agile, DMAIC, and Waterfall. Looking at Waterfall, this methodology lacks the ability to be flexible and iterative; therefore, we decided to consider other methods. DMAIC stands for Design, Measure, Analyze, Implement, and Control and is a data-driven nature. Due to this approach and inflexibility, we decided not to use DMAIC. Agile and Scrum are highly effective methodologies that focus on being flexible and team collaboration focused. However, we feel that Scrum suits us better because its structure uses sprints, which are effective in code-based projects and will allow us to focus on individual aspects of our project one at a time.

B. Solution Approach

We developed three primary solutions after evaluating the individual approaches below:

- Data Extraction Method
- Data Visualization Approach
- Dashboard Application

1) Solution I.: Initially, we decided to try and extract model data via the XMI output of a model from within Cameo Systems Modeler. While this approach simplified data extraction to one-click, the actual data within the XMI output was hard to read, unformulated, and impossible to parse. XMI outputs are designed for interoperability, but due to the complex nature of the software as well as the models that are created, the data representation was not human-readable and includes very few contextual clues about the element' functional roles in the model, making automation and parsing difficult, especially across different model types and languages.

One of the most important visualization approaches for this solution was to show semantic relationships between model elements, such as interface dependencies, allocation, signal flows, etc. The XMI output did not clearly expose these relationships in a way that could be reconstructed into a new dashboard. Identifiers for diagrams and their contents were entirely omitted from parts of the XMI output, reducing our ability to reconstruct a model within our dashboard. Diagram layouts, styling, and visual composition data—such as the placement of connectors, nodes, and any embedded images were encrypted within proprietary blocks. In parallel with the difficult data parsing, we attempted to visualize this semi-parsed data using Plotly's graphing libraries, specifically the network integration and sunburst plots for model structure exploration. Unfortunately, the resulting diagrams were visually overwhelming and prone to misinterpretations. Nodes within the graphs often lacked meaningful labels and relationships were indistinguishable without manual coding or heuristics, ultimately affecting the automated nature of the visualization approach.

When evaluating the extracting and visualization in early tests with non-technical users and stakeholders, the interface provided by Plotly-based graphs and tables were found to be non-intuitive and confusing. Users were unable to differentiate elements, navigate through the hierarchal structure of the model, and users would spend significant amounts of time learning about the model/what was going on within the model. In terms of dashboard application, we attempted to use a desktop application for our dashboard, which limited its accessibility across teams and required users to manage dependencies and runtime environments, adding unnecessary technical difficulty an already taxing approach to visualizing MBSE model data.

2) Solution II.: Following the limitations encountered in our initial XMI-based approach, we pursued a second approach that leveraged the native API of Cameo Systems Modeler to access and extract MBSE model content directly from within Cameo's environment. This method aimed to overcome the issues involved with the XMI extraction by directly querying model data through a scripting-based interface. At the time of development, our team had access to Cameo Systems Modeler 19.0 SP2, an older legacy version of the current software that is about six years old, posing several technical challenges. The available documentation for interacting with the API for version 19.0 SP2 was incomplete and inconsistent with modern scripting practices, many functions referenced in official community documentation for the API were either depreciated or unsupported due to the older nature of the software we were using. Due to this, we had to rely on a trial-and-error approach to pulling data out and reverse engineering the internal structure of the model through the API.

Furthermore, Cameo's internal scripting language uses Jython, which is an implementation of Python in a Java environment. This was necessary because the API documentation is written in Java only, so we could only pull data out and use the API with a Java-based environment. Unfortunately, the Jython interpreter only supports Python 2.7 and below, which lacks the modern syntax and libraries used in Python 3.0, which is the language we have learned to use Python with.

The extraction complexity was an issue as well, as although the API allowed us to access elements, there were often several layers within these models defining important relationships and critical attributes that would have to be visualized on the dashboard. Manuel traversal was required through inheritance trees and even when data was successfully pulled out, normalizing it into a dashboard-readable format produced maintainability issues. The extracted data was funneled into a Flask application via Python and rendered using HTML template to produce a prototype. This allowed us to experiment with a web-based interface, visualizing model summaries and basic diagrams viewable in any browser. This approach was promising as we were able to create interactive tables and a less overwhelming design as compared to approach one, it was still insufficient for non-technical stakeholders without considerable work being done post-processing.

This approach also utilized a progress web application (PWA) which is a web-application that can be used online as well as offline if a server is having connectivity issues. This approach also showed some promise, but the usability was constrained due to the secure nature of The Aerospace Corporations work as well as the incomplete nature of the underlying data pipeline. Despite these challenges, the second approach validated the feasibility of direct model access and real-time data transfer as well as a step in the right direction for visualizing data, as using HTML simplified model data and made the dashboard visually appealing.

3) Solution III.: Building on the lessons learned from our earlier prototypes, we shifted to a simpler and maintainable solution using Cameo Systems Modeler's built-in Report Wizard. This approach allowed us to move away from data extraction with the restrictive and proprietary scripting environment within Cameo using its API. Instead of relying on embedded scripting, we used the Velocity Template Language (VTL) to define templates that could export model elements, metadata, and relationships in a structured, machine-readable format.

This template-driven methodology enabled a clearer separation of concerns and allowed us to focus on formatting and preparing data for external processing without requiring direct interaction with Cameo's API at runtime.

This method was the configuration of custom VTL templates within Report Wizard. These templates enabled us to translate various SysML model elements such as blocks, requirements, ports, and diagrams into structured outputs like JSON. This exported data was then visualized on a standalone dashboard, leveraging Plotly/Dash for interactive charts and HTML/CSS for the overall interface design.

However, the development process posed high complexity. VTL differs significantly from standard programming languages, and its syntax and logic were unfamiliar to our team. In addition, VTL required a deep understanding of Cameo's internal metamodel, especially to extract nested or recursive relationships such as child components, nested relationships, and model structure. Constructing functional templates demanded extensive testing and experimentation, as basic debugging tools were not available.

One of the primary limitations we encountered was the lack of modern documentation. Although the Report Wizard is a central feature of Cameo Systems Modeler, the last meaningful update to its documentation was published during version 17.0.1, released nearly a decade ago. This gap left us without official references for newer model elements or advanced template syntax leading to lost time figuring out how to interact with a newer version of Cameo

Most built-in templates were geared toward generating human-readable documents (e.g., Word reports or PowerPoint presentations). There was little to no support for programmatic exports like HTML or JSON. As a result, we relied heavily on reverse engineering existing examples, forum discussions/GitHub Repositories, and trial-and-error development, as we would have to create a template, upload it to Cameo, run the template, see the output, and repeat the process until we got the information we wanted

Another challenge was model structure. The Report Wizard assumes certain structural conventions in the model. When these assumptions were violated (e.g., elements lacked stereotypes, missing relationships, or deeply nested packages), the output became inconsistent, unreadable, or even broke entirely, exporting information that wouldn't help visualize a model's information in a way for non-technical users to understand it.

The Report Wizard inherently produces a static snapshot of the model at the time of export. This means any model changes require a manual re-export, and the dashboard must be manually reloaded to reflect updates. This manual cycle is not ideal in high-velocity environments where stakeholders

need to interact with live or continuously updated models. Furthermore, integration into the Aerospace Corporation's pipeline was not straightforward, limiting opportunities for automation.

The front-end interface of the dashboard was developed as a web-based application, allowing for easy accessibility and universal compatibility. The app runs on a designated internal server and is fully accessible from any modern browser which eliminates the need for installation or specialized software on the user's device.

C. Chosen Solution

1) Data Extraction: After evaluating the benefits and limitations of several data extraction methods and visualization pipelines, we ultimately developed a hybrid solution that combines the Cameo Report Wizard and Groovy-based scripting via Cameo's internal API. These extraction tools feed into a Python-powered web application built using Plotly Dash, HTML/CSS, and structured data tables. This approach balances automation, reusability, and extensibility, while keeping the application lightweight and accessible across different user environments.

We used the Report Wizard in Cameo Systems Modeler to extract model data in a structured and repeatable way. Our configuration employed a custom Report Wizard template, written in Velocity Template Language (VTL), which programmatically pulls all requirements from any given MBSE model. The template recursively traverses each element starting from the root node, collecting all attached requirements. The extracted data is output in a formatted HTML table, which can be directly displayed within the dashboard.

To support this, we modified existing templates to match our desired export schema. These modifications included creating hierarchies through model elements and relationship chains that could later be visualized on the dashboard. The template processes each requirement by removing line breaks in names, generating dynamic table headers, and sorting the list by requirement ID. The result is a clean, readable HTML file that presents only the relevant requirement information—ideal for sharing with stakeholders.

To extend our capabilities further, we upgraded to Cameo/MagicDraw 2024x, which provided improved API access and updated documentation. We also transitioned from Jython to Groovy, a Java-based scripting language officially recommended for interacting with Cameo's internal API. Groovy offered better performance, improved compatibility, and a smoother scripting experience once the language was learned.

Our Groovy script serves as the main data extraction mechanism. It interacts directly with Cameo's backend to scan the full model hierarchy, collect detailed data on each diagram and its elements, and export visual snapshots. The output is organized into a structured folder containing a text file, a diagrams image folder, and an element image folder. For each diagram, the script logs metadata such as diagram type, ID, and model path, along with detailed information on associated model elements like signal events, change events, input/output pins, and relationships. These are both documented and exported as image files for dashboard visualization.

Additionally, the script tracks how diagrams are referenced across the model to support traceability and context awareness. It captures the full model structure by traversing the hierarchy from the top-level node down through each embedded element, preserving the logical layout as defined in Cameo.

After generating the structured text output, we convert it into JSON format to make it compatible with the dashboard's visualization engine. Using Python libraries such as json and os, we parse the text document to extract diagram metadata, build parent-child relationships, and reconstruct the full model hierarchy. This final structured JSON enables intuitive, interactive display of the model's content within the web dashboard.

2) Data Visualization : The dashboard was built with a Python backend powered by Dash and Flask, and a front end styled using HTML/CSS and Plotly components. To integrate data exported by the Report Wizard, we used Python's BeautifulSoup library to parse the generated HTML and embed it directly into the dashboard interface. For model structure and diagram navigation, we processed the JSON output from the API-driven extraction and created a diagram hierarchy table in HTML. This table captures the organizational layout of the model and its associated diagrams.

In addition, we used Python's OS library to match each diagram's unique ID in the JSON file with its corresponding image stored in the API output folder. This enabled the creation of interactive tables where users can view both the diagram visuals and their underlying metadata, all within the same interface.

3) Application: We chose a web-based application for this dashboard, as it aligned best with The Aerospace Corporation's system architecture and was the most practical solution for sharing the tool with clients who may lack the technical expertise to navigate complex MBSE models. By delivering the dashboard through a browser-accessible interface, we eliminate the need for users to install specialized software or operate Cameo Systems Modeler directly.

This format supports cross-platform compatibility, working seamlessly across Windows, macOS, and Linux operating systems. It can be hosted on either internal servers or cloud-based infrastructure with minimal setup, making deployment flexible and scalable. Multiple users can interact with the dashboard simultaneously, enabling real-time collaboration and broader accessibility.

Additionally, a web-based approach simplifies version control and updates—any changes made on the server are immediately reflected for all users without requiring local reinstallation. Ultimately, this delivery method transforms a traditionally technical and isolated engineering workflow into a collaborative, portable, and user-friendly experience.

D. Dashboard Features

The dashboard we created utilizes a variety of features which we have outlined below:

a) Diagram Navigation: A diagram heiracrchy table allows users to navigate between diagrams and is presented with all diagrams in a structured, collapsible format. The first column displays diagrams in hierarchical order and includes clickable links that direct users to the diagram viewing pages. Another column displays the path for accessing diagrams in Cameo, which is useful for those cross-referencing the source. Lastly, a "Nested Diagrams" column lists any diagrams embedded within others, and these too are clickable for seamless navigation .

b) Requirements Table: A crucial component within Model-Based Systems Engineering (MBSE) models is the set of system requirements. These requirements are typically established early in the development process and form the foundation for the model's structure and logic. They define system behavior and are critical for validating system performance against original goals.

To reflect this importance, we integrated a Requirements Table directly into the home page. This table displays all current requirements along with their unique identifiers and descriptions. A completion tracker was also added, enabling Project Managers to monitor progress on each requirement. By embedding this table into the dashboard, we enhance visibility and traceability, enabling users to easily track development progress and focus on pending items without needing to access Cameo.

c) AI Chatbot: As we explored additional features for the dashboard, we were drawn to the idea of an integrated chatbot. This assistant could help non-technical users interpret dashboard content more easily. If we could train a local LLM and build a knowledge base from extracted Cameo data, users could ask detailed questions about model content. Expanding this, Cameo models from across the organization could be added, turning the chatbot into an enterprise-wide MBSE assistant. To pursue this, we attempted to train a local LLM. Choosing the right hardware was essential. Our best option was a machine with an NVIDIA RTX 2080, which has 2,944 CUDA cores and 368 second-generation Tensor cores. The CUDA count was sufficient, but the older Tensor cores and 8 GB of GDDR6 VRAM were potential bottlenecks. Still, it was a capable setup for experimentation.

Training required a Linux environment to use NVIDIA CUDA, so we set up Windows Subsystem for Linux (WSL) and installed Ubuntu 24.04.2 LTS for its long-term support and stability. To simplify development, we used the WSL extension in Visual Studio Code, allowing us to run Python scripts directly. We then set up an Anaconda environment to manage packages and installed NVIDIA CUDA Driver 12.8.1, updating the GPU driver from 566.03 to 577.03. We also installed Conda cudatoolkit 11.8, which enables GPU support within Anaconda. At this point, our system was fully configured with WSL, Python scripting, a compatible driver stack, and GPU-ready development tools.

Next, we selected an LLM, built a knowledge base, and wrote the integration script. We used Hugging Face Hub, a platform similar to GitHub for ML models. Though we found no model trained specifically on Cameo data, we identified several that could process JSON input. We chose Mistral-7B-v0.1, a float16-optimized dialogue model. After registering on Hugging Face and getting an API key, we integrated the model into our script. Since it didn't natively support Cameo data, we converted the extracted data to JSON and stored it as vectors using a Faiss GPU (1.7.2) index for fast retrieval. The script loaded the Faiss index and model, then configured a prompt template, retrieval chain, and query loop. We were able to query the LLM, but performance was slow, largely due to the model requiring fine-tuning—something our hardware could not handle effectively.

Two major limitations emerged: our delayed Cameo access and hardware constraints. Fine-tuning the model would require more time and GPU power than we had. With the deadline approaching, we had to choose between continuing with a slow local model or switching to a faster LLM API. We chose to pursue an API solution to enable a smoother, real-time demo. Cost was a concern, so we selected Google's Gemini API, which provides free credits and access to Google's LLM models.

We evaluated four Gemini models—2.0 Flash Lite, 2.0 Flash, 2.5 Flash Preview, and 2.5 Pro Preview—using a decision matrix. Our highest priority was latency (weight 0.4), followed by size and scalability (0.25 each), and output type (0.1). The model 2.0 Flash Lite scored 4.6/5, making it the best fit. With the model selected and API integrated, we were ready for implementation.

To integrate the chatbot, we added a chatbot icon to the dashboard, initialized the Google Gemini API, and configured a modal for response formatting. We also added callbacks to manage user queries. The result was a working proof-of-concept chatbot, embedded directly into the dashboard, providing real-time assistance through a streamlined interface.

d) Help Page: Since this dashboard is designed for non-technical stakeholders, we included a dedicated help page to support onboarding and independent exploration. This page features the dashboard user guide, which offers step-by-step usage instructions, best practices, and tips for navigating the tool. Background information on SysML is also provided, including its purpose and core components. This section is intended to familiarize less experienced users with the language that underpins MBSE modeling in Cameo. Finally, there is a Frequently Asked Questions (FAQ) section. This interactive feature reveals answers when a question is clicked. The initial set of questions was derived from feedback gathered during our System Usability Scale (SUS) survey, ensuring relevance to actual user concerns.

IV. Results

To measure our impact on the usability of the system and how easily it portrays the information from Cameo in a new environment. To do this, we used the System Usability Scale (SUS) score that we discussed before. We surveyed 15 non-technical users with both the currently used system, Cameo, and the dashboard we developed. The survey consisted of six

questions that asked the user to answer the following on a five-point scale, 1 being "Strongly disagree" and 5 being "Strongly agree":

1) I think I would like to use this tool frequently.
2) I found the tool easy to use.
3) I think I could use this tool without the support of a technical person.
4) I found the various functions in this tool were well integrated.
5) I would imagine that most people would learn to use this tool very quickly.
6) I felt very confident using the tool.

The survey participants were instructed to use both systems for approximately 10 to 15 minutes before completing the survey. The baseline survey completed after using Cameo resulted in a SUS score of 1.66/5. The survey after using our dashboard had a SUS score of 4.64/5. This represents an improvement of 180% in usability. This is crucial in our project as increased usability creates a more efficient system of sending MBSE models to clients.

V. Impact

Most of the financial impact from our solution stems from the reduction in the number of Cameo Systems Modeler licenses required by both The Aerospace Corporation and their clients. Based on discussions with Aerospace stakeholders, we estimated that approximately 35 licenses could be eliminated through the implementation of the interactive dashboard. Given the high cost of each license, this reduction alone represents a significant recurring cost savings.

In addition to direct licensing savings, we identified three key intangible benefits that contribute to long-term financial and operational efficiency. The first is improved communication between Aerospace and its clients, particularly between technically experienced team members and those with limited or no technical background. The dashboard simplifies how model data is accessed and visualized, which reduces the need for repeated clarification or explanation during cross-functional collaboration.

Secondly, the system significantly reduces onboarding time for new team members who need access to MBSE data. The streamlined interface and user-friendly navigation make it easier for new hires or external partners to begin working with MBSE models without extensive training. Clients accessing the models benefit similarly, as the dashboard lowers the learning curve and shortens the time required to understand the material.

Lastly, because the software we developed is compatible with all MBSE models, it serves as a reusable tool. This reduces the need for modifying models on a case-by-case basis before sharing them with clients. Over time, this leads to measurable time savings for engineering teams and ensures consistent presentation of model content.

The combination of reduced licensing costs and these intangible benefits results in an estimated cost savings of $132,000 per year.

VI. Future Recommendations

A. Interactive Diagram Enhancement

Currently, diagrams are displayed as static images. Future teams could explore ways to make these diagrams interactive—allowing users to click on elements to reveal tooltips, metadata, or linked diagrams. This would significantly increase usability and enable real-time model navigation, especially for larger, more complex systems.

B. Real-Time Data Sync and Automation

The dashboard currently requires manual export and reload to reflect model changes. We recommend developing an automated data pipeline that syncs the latest model state with the dashboard, potentially triggered by commits in a version control system (e.g., GitLab). This would support continuous integration workflows and enable dynamic updates for live reviews or audits.

C. Security and Access Control

As the dashboard matures toward production, security becomes paramount. We suggest implementing user authentication, access roles, and integration with The Aerospace Corporation's internal identity and access management (IAM) system. This would ensure model access is protected and compliant with internal security protocols.

D. Model Validation and Feedback Schools

Introducing validation workflows directly within the dashboard would allow engineers to review and annotate model sections, flag issues, or confirm competencies.

E. Mobile-Friendly and Offline Capabilities

Although the dashboard is browser accessible, optimizing its layout for responsiveness for mobile devices would further expand accessibility, as well as making it a progressive web application so it can be run offline as well as online.

F. Locally Trained and Running LLM

As we have mentioned previously, having an LLM that is trained for the application it is being used for would drastically improve the dashboard. Additionally, a custom knowledge base can be created that stores all the extracted and formatted Cameo data. This would allow the user to ask specific questions on the Cameo models and as stated before it could be developed into an MBSE assistant for the entire company.

VII. Conclusion

This project addressed a critical challenge faced by The Aerospace Corporation which was enabling non-technical stakeholders to engage meaningfully with complex MBSE models traditionally locked behind proprietary, expensive, and technical software like Cameo Systems Modeler. Our primary objectives were to remove software access barriers and reduce operational costs, all while improving usability and scalability.

Through multiple design iterations, we evaluated various data extraction and visualization approaches including XMI parsing and API integration, before deciding on a hybrid solution combining Cameo's Report Wizard with Groovy-based API scripting. This allowed for structured, repeatable data extraction and visualization of diagrams, requirements, and system hierarchies in a Python-based web dashboard built using Plotly Dash and HTML/CSS.

The final dashboard is fully browser-accessible and cross-platform compatible which eliminates the need for users to install or understand Cameo. Key features include a searchable requirements table, interactive diagram navigation, side-by-side viewing, and an integrated AI chatbot powered by Google's Gemini API. A dedicated help page further supports onboarding and navigation for non-technical users.

Usability testing using the System Usability Scale (SUS) showed a significant improvement: scores increased from 1.66 to 4.64 out of 5, representing a 180% gain in perceived usability among surveyed users. This validated our approach to simplifying MBSE accessibility.

Financially, our solution is projected to reduce the need for up to 35 Cameo licenses, resulting in estimated annual savings of $132,000. Additionally, the dashboard improves communication, reduces onboarding time, and provides a reusable interface compatible with models of any size or complexity.

Overall, this project successfully transformed a traditionally technical and complex engineering process into a collaborative, intuitive, and cost-effective system that empowers a broader range of users to interact with MBSE models.