Implementing the Digital Thread - A Proof-of-Concept

Zayn Roohi∗, Gabriel Fronk†, Sebastian Abelezele‡, Ruby Al Fawares§, Olivia J. Pinon Fischer¶, Aroua Gharbi‖, and Dimitri N. Mavris∗∗

Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA, U.S.A.

Juan Oroz††

Georgia Tech Research Institute, Huntsville, AL, U.S.A.

Melissa J. Petersen‡‡

Rolls-Royce North American Technologies, Inc. (LibertyWorks), Indianapolis, IN, U.S.A.

Alexander Karl§§, John Matlik¶¶, Bryan Schwering∗∗∗

Rolls-Royce Corporation, Indianapolis, IN, U.S.A.

Current engineering processes are heavily document-centric, which can add time and cost to projects, limit data accessibility, and make it difficult to actively manage models and data consistency throughout the lifecycle of a product. Additionally, traditional data siloes limit data accessibility across departments. Similar issues exist with tools: departments use software with different standards and formats, making it time-consuming and difficult to accurately propagate changes and requirements throughout. Aerospace projects and vehicles are also often a level of magnitude more complex than products developed in other industries, requiring the coupling of multiple disciplines, which intensifies these problems. Digital Engineering and Model-Based Systems Engineering (MBSE) provide the context, methodologies and tools to address some of the aforementioned challenges. In particular, this paper presents the development and implementation of a Digital Thread proof-of-concept for a minimum viable product. In doing so this research demonstrates solutions to the challenges of data acquisition and management, model and data connectivity, tool and platform integration, eventually leading to the realization of an authoritative source of truth across the product’s lifecycle. Additionally, this research highlights some of the key benefits brought about by the Digital Thread, which include increased collaboration and communication, managed consistency across models and data, as well as the ability to conduct model verification, validation, and calibration - an important tenet of MBSE.

I. Introduction

Current engineering methods are generally heavily document-centric, a term that refers to how information on designs, requirements, and other key data is stored in individual documents. When updates or changes are made, they propagate out to other teams and individuals only by users manually changing or updating their documents. In recent years, however, engineering firms have slowly been shifting to Model-Based Systems Engineering (MBSE). MBSE replaces the disparate documents with a centralized model through which information can be gathered or exported.
The model itself contains all of the needed information, so when changes are made, the model updates in real-time, everywhere, for all users [1]. However, many difficulties remain to be addressed, calling for a greater integration of digital technologies. These can be categorized in terms of: challenges related to data, challenges related to tools or standards, and challenges related to consistency management.

In most organizations, data is stored locally or on network servers by the department(s) or user(s) from which it originates. Access is generally restricted to those in the department, a situation known as data siloing [2]. Data silos limit data visibility and accessibility to others who may need it, both within and outside of the organization. For example, design engineers traditionally do not have access to the operational or manufacturing data related to a product they designed. Very similar issues exist for tools and processes. Different departments and groups use a diverse set of software, tools, and platforms, which have not been designed or developed to integrate with one another. Standards often differ across organizational boundaries, meaning that the formatting or quality of data for one department may be quite different from another. One final issue is a common lack of consistency management. Consistency management is critical as it helps ensure that requirements are consistent with stakeholder needs, the design is consistent with these requirements, simulations and models are consistent with the design and requirements, and so on [3].

A. Digital Thread: Definition and Value

Digital Engineering provides the tools, methods and processes to help address the aforementioned challenges. One important and tangible component of Digital Engineering is the Digital Twin, which is defined as a "virtual representation of a connected physical asset" [4]. Key to the connection between the physical and digital worlds is the Digital Thread, which is defined by the Defense Acquisition University as "An analytical framework that seamlessly expedites the controlled interplay of authoritative data, information and knowledge to inform decision makers throughout a system’s life cycle by providing the capability to access, integrate and transform disparate data into actionable information" [5]. The Digital Thread provides a number of benefits. Those benefits, which are discussed in detail in [6], are summarized below:

- **Consistency Management**: The Digital Thread is key to one’s ability to actively manage the consistencies (system requirements vs. stakeholder needs, virtual simulations vs. empirical measurements, etc.) between the various information constructs (data, models, etc.) the Digital Thread encapsulates [7].
- **Bi-directional traceability**: The Digital Thread allows for data and models to be linked and traced across and along the lifecycle and the various domains involved [6]. Hence, by enabling information to be passed backwards and forwards throughout a product’s lifecycle, the Digital Thread ensures that engineers can access operational and manufacturing data, or users of the product can access manufacturing and design data. This increases information flow throughout departments and organizations, enabling higher levels of communication and collaboration [8].
- **Communication & Collaboration**: The Digital Thread enables increased communication and collaboration by enabling easier and quicker access to data across organizational borders. This can help set organizational or industry-wide standards, which allows for even easier collaboration [9].
- **Analytics**: The increased accessibility to data and its integration with models provide opportunities for the development of analytical capabilities (descriptive, diagnostics, predictive and prescriptive), which in turn inform decision-making.
- **Knowledge & Model Reuse**: The Digital Thread allows for data, models, and knowledge to be stored and properly indexed by means of data management tools. This in turn allows for current as well as future designers, engineers, users to search, retrieve, and access the data or knowledge, and then integrate it into their project, as needed [10–12].
- **Model Support**: A final important benefit of the Digital Thread is its ability to support model-based systems engineering and the calibration, verification and validation of Digital Twins. These three processes enable and guide the development of valid Digital Twins as well as help quantify their accuracy and the confidence in their predictions.

B. Existing Implementations

A number of organizations have successfully implemented elements of the Digital Thread in recent years. Boeing, for example, implemented their first Digital Thread during the development of the Boeing 777 in 1990 [13]. This first instantiation of a Digital Thread was used to send errors from physical parts directly to design, manufacturing, and other relevant engineers. Most recently, Digital Twins are created for every unique aircraft manufactured, with the
Digital Thread being used to connect the physical and digital worlds [14]. As a result of implementing a Digital Thread, defective parts rarely reach the assembly line, and when errors occur, engineers are rapidly able to identify the root cause of a problem. Additional benefits include a 25% reduction in development time, 50% reduction in software costs, 80% reduction in assembly hours, and 200 fewer people involved [14]. However, one of the largest roadblocks to implementation is often the sharing of data with other entities, such as between original equipment manufacturers (OEMs), engine manufacturers, and suppliers [13].

Most recently, the U.S. Air Force’s Next Generation Air Dominance (NGAD) program has attempted to leverage the Digital Thread to decrease time and cost between aircraft generations by ingesting data from physical prototypes and digital tests into a digital design environment. While very few public announcements have been made on the program, the Air Force has stated that this setup has allowed for digital models and simulations that are of high-enough fidelity to be used to reduce physical prototyping at certain stages of development, where previously required.

The automotive industry has generally led the adoption of digital technologies, and this has been no different with the adoption of the Digital Thread. The fully digital nature of electric cars facilitates the implementation of these digital technologies, leading Tesla to become an industry leader in that area. Indeed, Tesla creates a Digital Twin of every car it manufactures and sells. The Digital Twins are then continuously updated with physical sensors and driving data using the Digital Thread. As a result, engineers are then able to incorporate this real-world, operational data into their design environment, use it to optimize performance, and then propagate it back to the cars through software updates [15].

C. Implementation Challenges

While a number of organizations have successfully implemented elements of the Digital Thread in recent years, this has not been without challenges and hurdles, as discussed below:

- **Implementation costs:** One of the first major hurdles reported is the cost of creating and setting up the Digital Thread. A key benefit of the Digital Thread is that it will eventually lead to lower costs and reduced development time. However, the initial upfront cost of creating the Digital Thread is prohibitively high for many organizations. One study looking at a large-scale implementation of the Digital Thread for the U.S. Air Force estimated that it would cost between $1 to $2 Trillion and take between 100 to 250 years to fully setup. The yearly maintenance costs would be around $100 Billion. Instead of dedicating the time and funding to completely setup a full Digital Thread, the authors recommended creating smaller, more focused Digital Threads that could realistically be achieved in short time-spans and for less money [16].

- **Data handling:** Another challenge concerns the quantity of data to be managed by the Digital Thread. Referencing the aforementioned study, implementing the Digital Thread for the Air Force would require computers and processors that could handle roughly 1-Petabyte of data and material from aircraft. This task would require computers close to 30 times as powerful as today’s computers [16]. Addressing this issue would require limiting the data identification and collection efforts to the data that is strictly required for specific systems of interest as opposed to all data from all systems.

- **Data sensitivity & confidentiality:** To reap the full benefits of the Digital Thread requires that data be shared across organizational boundaries, e.g. between OEMs, suppliers, etc. To ensure the protection of proprietary and/or classified data and prevent corporate espionage, digital ecosystem platforms and data governance processes need to be developed that provide access to authorized persons only and ensure data integrity [17].

- **Data standards & format:** The general lack of standards regarding formats, files, or data exchanges impedes collaboration and represents yet another challenge to the implementation of the Digital Thread. Indeed data, knowledge, and models exist in a wide range of formats, not all compatible with the tools and platforms that need to ingest them. The development of industry-wide, open standards is needed to ensure data and model interoperability within the Digital Thread [17].

The objective of the present paper is to demonstrate the value, benefits, and challenges of developing and implementing a Digital Thread in the context of a minimum viable product, in this case a small Unmanned Aerial Vehicle (UAV). As such, this paper discusses how data and models (system model and analytical models) from the “As Designed”, “As Built”, and “As Used” phases of the product lifecycle are knitted together to represent an authoritative source of truth. In particular, this paper demonstrates the feasibility and appropriateness of the approach and technologies selected and reports on lessons learned regarding integration and scalability to larger, more complex systems.

For this purpose this work builds on existing research conducted on the topic of Digital Enterprise across the lifecycle [18]. In particular, it leverages the minimum viable product (Figure 1) and augments some of the models
developed (Figure 2) as part of the effort described in [18].

![Fig. 1 2016 Design Build Fly (DBF) vehicle](image)

Section II presents the methodology, Section III discusses its implementation and Section IV presents the results from the execution of the Digital Thread. Finally, Section V summarizes both benefits and challenges, and discusses lessons learned and avenues for future work.

II. Methodology

Section II.A first discusses the proof-of-concept the methodology is implemented on. Section II.B then discusses the foundations for the Digital Thread developed as part of this effort. Section II.B.3 presents the concept of operation of interest and Section II.B.4 introduces the proposed approach to the implementation of the Digital Thread.

A. Proof-of-Concept

As mentioned, this effort builds on the models developed as part of [18] and captured in Figure 2. For the purpose of this research, a battery model is used as a proof-of-concept to demonstrate how the Digital Thread supports 1) the bi-directionality of data, 2) the updating of models with the appropriate functional data, and 3) calibration, verification and validation activities using simulated battery data as well as actual data from the aircraft in flight.
B. Foundations for the Digital Thread

1. Simulation Infrastructure Standard

Prior to developing the Digital Thread, simulation standards were reviewed in order to guide the architecting of the Digital Thread and the reuse of simulations. In the context of this research the Digital Thread is built around the industry simulation standard known as the Department of Defense High Level Architecture (HLA). The HLA is a simulation infrastructure standard that supports interoperability and reuse of simulations [19]. Many aspects of the HLA equate well to the Digital Thread principle, meaning that the HLA provides a good foundation for designing the Digital Thread. This standard consists of an object model template, an interface specification, a runtime infrastructure, and additional rules governing how the various entities in the enterprise operate with one another (Figure 3). The object model template describes all of the objects (models, files, personnel, etc.) involved in the enterprise, including their attributes and relationships. The interface specification, as its name suggests, controls the interface with each of the objects and the runtime infrastructure. The runtime infrastructure (RTI) coordinates operations between models, manages data exchange, and designates time and ownership management of data and files. The RTI acts as the command center of the architecture, managing when codes are executed and models updated. The various roles designated in the HLA are used as the backbone for architecting the Digital Thread.
2. Selection of Digital Thread Components

Ideally, the Digital Thread would be a single tool that provides all of the functionality needed for the interplay of data and knowledge. In reality, no such holistic tool currently exists, partially due to the fact that there is no industry standard for the development of Digital Threads. For lack of a single tool, many programs are selected to work in tandem to provide all of the capabilities desired for this Digital Thread implementation. The definition of Digital Thread provided in Section I.A can be broken down to identify requirements for the Digital Thread. From there, tools can be qualitatively benchmarked to identify the ones that are more likely to help meet each of these requirements. For readability purposes this definition is repeated below:

“An... analytical framework that seamlessly expedites the controlled interplay of authoritative data, information and knowledge to inform decision makers throughout a system’s life cycle by providing the capability to access, integrate and transform disparate data into actionable information.”

The definition of the Digital Thread as an “analytical framework” calls for means to represent the relationships within and across models from different sources. To that end, the graph database engine Neo4j [20] was selected. Its framework involves nodes (which are elements that represent each entity), relationships (which connect nodes), labels (which groups nodes), and properties (which are the name-value pairs that adds qualities to nodes and relationships). This graph database also enables the ability to control the “interplay of authoritative data” by introducing version control. Related to this concept of controlled interplay of authoritative data is the ability to integrate disparate data, as mentioned in the definition. These concepts correspond respectively to the RTI and interface specification described in the HLA standard. For the purpose of this work, these capabilities are fulfilled using Phoenix Integration’s ModelCenter [21].

The need to “access...disparate data” requires repositories in which data and models can be stored. Microsoft’s SQL database services provide a good flight and analysis data repository, while GitHub functions well as a blob storage for models and other unstructured data. Ultimately, the Digital Thread must “transform disparate data into actionable information”. This in part means that the Digital Thread must support the verification, calibration and verification models of real assets, as mentioned previously. The various models and analysis scripts (MagicDraw, MATLAB, etc.) developed as part of [18] provide this important information. Finally, in order to “inform decision makers” the data and analysis results need to visualized. This is achieved using dashboards created in Tableau [22].

3. Concept of Operation

With a clearer vision of how the Digital Thread is to be architected, understanding how the Digital Thread functions in the larger context of the enterprise brings a clearer understanding to how it should be implemented. This concept of operation is illustrated in Figure [4].
The cloud storage allows for the data to be transferred efficiently, but the data format must be transformed as well. The models used by the various groups may not have exact mappings to one another, and so the Digital Thread provides a common ontology so that the various disciplines can communicate and collaborate efficiently. The graph database, due to its flexibility in representing various kinds of information, provides this common language. It ensures that all models are up-to-date with the most recent analysis and data. The graph database also provides important metadata for data tracking. The analysis of the various groups can also be shared easily with dashboards.

Once the design and manufacturing processes are complete, the asset is ready for operation. The base data that this analysis executes on comes from constantly updated flight data collected from real-world use of the asset. Throughout its lifetime, the asset provides pertinent data to the Digital Thread, allowing for the models to be (re)-calibrated and updated, as needed. The flight data can then be used to inform decisions to be made in the “As Designed” and “As Built” phases of the life cycle. The bi-directionality of data across all phases of the lifecycle represents one of the main benefits of the Digital Thread.

4. Proposed Approach

The process of developing the Digital Thread is broken down into three broad steps, as illustrated in Figure 4:

1) Visualizing the relationships across the Digital Thread elements (e.g. models, people, data, metadata).
2) Creating the Digital Thread through the integration of models and data.
3) Executing the Digital Thread, demonstrating how analyses are run and data transferred across the physical and virtual worlds for the purpose of model verification, calibration and validation.
III. Implementation

This section describes in detail the steps taken towards the realization and execution of the Digital Thread for the minimum viable product of interest.

A. Step 1: Visualizing the Digital Thread Elements

The first step in implementing the Digital Thread consists in visualizing the many relationships that exist between models, data, and people. The ability to visualize those relationships helps to provide a quick assessment of the 'completeness' of the Digital Thread and whether models and data are connected in a meaningful way.

In selecting a tool to allow for the visualization of the Digital Thread, two key qualities were desired. These are the ability to query relationships and the ability to actually connect the various elements of the Digital Thread in addition to visualizing their relationships. When choosing a semantic or graph database to perform this function, it was found that some semantic databases were limited in the range of elements that could be modeled. Ultimately, Neo4j was selected as the tool of choice to visualize the Digital Thread due to its flexibility and ability to capture different kind of information. Neo4j is a graph database which uses nodes and edges to represent data. The nodes represent the various entities in the enterprise; the edges represent relationships between nodes. Each node and edge contain labels and properties that make metadata about the entities accessible. Examples of metadata represented include the file path of data, the input data needed, where the input is collected from, the output data and where it is required, the executable environment of a model, the contact information of the person who developed the model or collected the data, etc. Neo4j does have a limitation in that it cannot actually create connections between models and requires manual input by a user to update.

The Neo4j server for this Digital Thread implementation contains nodes for each person in the project, system level requirements, design and manufacturing models, repositories, modeling and simulation codes, and the physical minimal viable product with its flight data. Figure 5 shows some of these models. Neo4j’s main strength lies in its ability to query data according to the life cycle phase, data type, etc. To augment the utility of Neo4j, the concept of a Gold Layer is introduced, where the Gold Layer defines a model as being the most up-to-date version, ensuring that no data being used is obsolete. The Gold Layer supports consistency management in the Digital Thread.
B. Step 2: Storing Data

Two main types of data need to be captured in the Digital Thread: structured and unstructured data. Structured data is typically categorized as quantitative data, highly organized and easily decipherable by machine learning. Unstructured data, on the other hand, is typically categorized as qualitative data, and data which cannot be processed and analyzed via conventional data tools and methods [23].

1. Structured Data Storage

A tool was selected to store the data that captured the analysis parameters and metadata in an accessible database. The following capabilities/attributes were evaluated when selecting such tool:

- \( C_1 \) Disparate Data Storage
- \( C_2 \) Affordability
- \( C_3 \) Interplay with Integration Platform
- \( C_4 \) Structured Data Storage and Query
- \( C_5 \) Data Accessibility by User
- \( C_6 \) Data Security
- \( C_7 \) Community Acceptance

The tools evaluated were Microsoft (MS) Azure SQL and MS SQL Server 2019. MS SQL Server 2019 was eventually selected.

2. Unstructured Data Storage

A similar approach was followed for the selection of the unstructured data files storage. The same capabilities/criteria as above were evaluated. The tools compared were Microsoft Blob Storage, GitHub and BitBucket. Github was eventually chosen.

Node.js, a JavaScript run-time built on Chrome’s V8 JavaScript engine [24], is required to extract the information from the MS Blob Storage. A guide is provided by Microsoft to download blobs stored in Azure to a local drive [25]. This allows the user to program Node.js to download the latest baseline analysis, data sets, etc. from the Cloud.

While a Microsoft Azure SQL server, databases, and blob storage were initially set up for the purpose of this research, the costs associated with this service were too substantial and an alternative solution had to be found. As a result, to simulate the results that an Azure cloud solution would provide, a local MS SQL server was set up along with a free student version of GitHub used not only as a blob storage, but also as a file versioning tool.

C. Step 3: Connecting the Elements of the Digital Thread

The ability to connect different types of tools and models is key to the realization of the Digital Thread. To that end a number of tools/platforms were benchmarked against the following set of capabilities/criteria:
The tools/platforms compared were Syndeia, eQube, MagicDraw, ModelCenter, Ansys Connect. ModelCenter was eventually chosen. ModelCenter allows for the integration and execution of any modeling and simulation tool. In the context of this research ModelCenter is used to integrate SysML, CAD and simulation models. In addition, ModelCenter can execute and transfer data among connected models. ModelCenter has several limitations, however, one of which being that it does not enable consistency management across models. It also does not have the capability to graphically display the connections in the Digital Thread.

Several MATLAB models were integrated in ModelCenter Integrate. Figure 7 illustrates the two MATLAB models that drive the analysis portion of the Digital Thread. The analysis model is connected to a MATLAB function file that takes the analysis outputs and inserts them into a SQL database.

![Fig. 7 ModelCenter Integrate Analysis Set-Up](image)

Figure 8 illustrates how ModelCenter MBSE natively uses the value properties of the battery model and links them to the analysis created in ModelCenter Integrate, increasing the efficiency with which requirements can be verified in ModelCenter MBSE.

![Fig. 8 ModelCenter MBSE Server Analysis Set-Up](image)
D. Step 4: Modeling the Digital Thread
The Digital Thread was modeled using SysML in MagicDraw and leveraged the system model developed as part of [18]. Modeling the Digital Thread in SysML provides numerous advantages and in particular allowed the team to model the Digital Thread in a hierarchical manner. As a result, the team was able to rapidly observe and analyze where tool gaps existed within the Digital Thread. The team was also able to visualize the “Item Flows” between software systems.

1. Enabler: Object-Oriented Systems Engineering Method
As a requirement for accommodating multiple aircraft in the system, a unique structure was used in SysML. The model structure found in INCOSE’s Object-Oriented System Engineering Method (OOSEM) [26] was closely followed. OOSEM leverages object-oriented concepts and helps architect flexible and extensible systems.

![OOSEM Pyramid and Activities](image)

Figure 10 illustrates how OOSEM can be leveraged to develop a package structure that allows for the Digital Thread system to be expanded over time. The Digital Thread SysML model was closely modeled to Sanford Friedenthal and Chris Oster’s ‘Architecting with Spacecraft’ MagicDraw example [27]. Within the Digital Thread model is the Digital Engineering Framework, which follows the same OOSEM model structure as the Digital Thread model in which it is encapsulated. Within the Digital Thread SysML model are the project usages of the Aircraft model and the ‘ModelCenterAnalyses’ Database model.
At a high level, the Digital Thread can be decomposed into three major elements, shown in Figure 11, which are the production aircraft, the stakeholders and the digital engineering framework system. The stakeholders’ group can be decomposed into seven categories, which include the analytics, test, production, technology, engineering, configuration management and operations/support teams. Each stakeholder groups’ roles are unique, and their roles depend on what information and data they provide to the Digital Thread.
2. Operational Objectives

To explain how the Digital Thread operates, the model is broken down into operational and process objectives use cases. The operational objective explains how the user leverages the Digital Thread and what outputs/products it receives from the Digital Thread. Figure 12 illustrates the major use cases within that make up the enterprise operation activity. This includes retrieving flight data from the blob storage, executing the digital framework, selecting data inputs, and creating dashboard/analytic platforms for stakeholders.

Retrieving flight data: To perform the activity of retrieving flight data from the blob storage, Node.js and the MS Azure Blob Storage were used. Figure 13 illustrates the swim lanes and activities necessary to perform this capability in the Digital Thread. From both the use case and activity diagram, the stakeholder has no interaction with this capability. That is because Node.js is programmed to pull from the latest versioned files from the blob storage, while ensuring that only the latest information is being used.

Selecting input data: A second use case that is used in the operational objective is the selection of input data provided by the stakeholders. To that end, ModelCenter Analysis Server is used for the selection of execution plans and analyses. If the execution plan and analyses are correctly set-up in the analysis server, the stakeholder is then able to input the correct parameters to successfully execute an analysis.

Executing the digital engineering framework: The third use case for the operational objective is executing the digital engineering framework.
engineering framework. This includes executing MBSE tools by the stakeholders, ModelCenter executing a particular analysis, the SQL database storing the analysis data after execution, checking outputs against requirements, and updating the Neo4j Digital Thread model.

*Creating dashboards and analytic platforms for stakeholders:* The last capability as part of the operational objective activities consists in creating dashboards and analytic platforms for stakeholders. This involves Tableau retrieving data from the MS SQL database and processing the data to create dashboards for analytic purposes.

The enterprise operation activity diagram in Figure 14 illustrates the logical activities that are necessary to run an analysis in the Digital Thread and to create informative dashboards and analytic for multiple stakeholders.

![Fig. 14 Enterprise Operation Activity Diagram](image)

3. Executing the Digital Thread

Executing the digital engineering framework requires several capabilities from several software systems, inputs, and data from the stakeholders. Figure 15 illustrates the necessary actions including activities that might "extend" the analysis of a particular model. The activities that are required to execute the Digital Thread are executing MBSE tools, executing the analysis, storing the analysis data in the database, checking the outputs against requirements and updating Neo4j.

One of the first steps to execute the digital engineering framework is to execute the MBSE tools. Figure 16 illustrates the necessary activities required in order to execute all the necessary software tools to perform an analysis in the Digital Thread. This includes executing the analysis server, executing MagicDraw, logging into TeamWork Cloud (TWC), and executing ModelCenter MBSE from the MagicDraw ‘Tools’ pull down menu.

The second step in executing the digital engineering framework consists in executing the analysis. When analyses are integrated with ModelCenter, ModelCenter acts like a conductor of the software which executes the analyses. ModelCenter commands MATLAB to execute a particular MATLAB .m file. MATLAB determines that the analysis requires the selection of a data file from the stakeholder conducting the analysis. Once a file is selected, the battery model then runs through the rest of the analysis. Once MATLAB finishes executing the desired analysis, it stores the output values from the analysis, unless it encounters an error within the analysis. This is realized by ModelCenter commanding MATLAB to run the .m function file to begin and process the storage of data in a SQL database. MATLAB first establishes a connection with the SQL database to verify credentials. Once that connection has been established, MATLAB creates a table that will then be translated into a SQL table. The MATLAB file then utilizes the Database Toolbox to convert that table and send it to the SQL database for storage.

A unique feature that ModelCenter possesses is the capability to verify requirement threshold/objective values by comparing them with the output values of the analysis. For this, ModelCenter MBSE first requests a requirements ID and associated value property from MagicDraw. MagicDraw sends the requested information to ModelCenter MBSE. ModelCenter MBSE receives the requested information from MagicDraw and compares the analysis output against threshold/objective values of the requirement. The last step is displaying the results to stakeholders, with either a check or an ‘x’ next to the requirements ID. An example of these results can be found in Section IV.

Ideally, to visualize the Digital Thread and its changes, metadata would be captured in the SQL database every time a new analysis is executed. The execution of an analysis would create several metadata parameters that would in
theory make connections to the Digital Thread in Neo4j. Figure 17 illustrates both the automatic and manual methods to updating Neo4j. If the system is set up to automatically update from the SQL databases, Neo4j would request access to the system by authenticating into the SQL server. Once given permission, Neo4j would be able to receive the tables and its data into the Neo4j environment.
4. Process Objectives

A process objective was modeled to differentiate between the stakeholders executing an analysis and providing other types of data or changes to the baseline models to the digital engineering framework. Figure 18 illustrates the processes of committing an analysis by the analytics team, retrieving battery flight data by the test team, updating and modifying requirements by the engineering team and verifying the baseline by the configuration management team. These processes are further discussed below.
Committing an analysis: After a baseline has been approved by all stakeholders, the analysis team can commit a change to the blob storage to either modify an existing file or a new analysis. To do so, the analysis team would first authenticate on the blob storage server and select the file to be modified or added and sent to the blob storage. The blob storage would then receive it and confirm to the analysis team that it has been properly baselined and versioned.

Retrieving battery flight data: Another process objective is retrieving battery flight data from the aircraft. This process involves the pilot first connecting a USB cable to both the Pixhawk 4 and the computer. The pilot would then execute the Pixhawk software to retrieve the data and convert the raw data retrieved from the Pixhawk to either a .csv or .mat file.

Updating and Modifying requirements: Another part of the process objective is the capability to update and modify the requirements. When the engineering team receives approval to make changes to the requirements in the system model, a baseline set of requirements can be committed to the model. The engineering team would first log in to TWC and open the latest committed version of the system model. Ideally, the engineering team would use a ReqIF file to update the new baselined requirements. Once the new requirements have been accepted by the system model, the new and/or modified requirements will be highlighted. The engineering team then commits the changes to the model to TWC and then checks if the new branch has been created from the modification to the model.

Verifying the baseline: Baseline verification is necessary to ensure that the Digital Thread is up to date, and that it is the sole source of truth. To do so, the configuration management team would authenticate their identity in the blob storage database. The blob storage would then generate a report of the baseline analyses files. The configuration management team would then verify that the blob storage has the latest baselined files ready for the appropriate stakeholders to access.

5. Data Exchange in the Digital Thread

As discussed in Section III.B, the Digital Thread is used to transfer both structured and unstructured data. Figure 19 illustrates the value types captured in the process of operational and process objectives. These value and data types can be visualized in SysML as ‘items’ flowing through the Digital Thread system.
At a high level, there is an exchange of data between the stakeholders, the digital engineering framework, and the production aircraft. Figure 20 illustrates the items flowing within the Digital Thread. It can be observed that most of the data being exchanged is from the stakeholders to the digital engineering framework. This data mostly consists of unstructured data such as flight data, analysis files and SysML files.

Once the data has been transferred to the digital engineering framework, it can then be passed to the digital engineering framework subsystems, which include the digital backbone and terminals. The digital backbone is primarily made up of the blob storage, TWC, and the SQL Server. Figure 21 illustrates the data flowing between the digital backbone and terminals. Within the digital engineering framework is where the system begins to transfer structured data. This is mostly done through a connection between the SQL server, ModelCenter and Neo4j.
MagicDraw can connect to a SQL server and model a selected SQL database in SysML. The modeling of the SQL database was done in a separate mdzip file for modularity of expansion of the database of the whole Digital Thread. For the purpose of this research, six values were being fed into the battery analyses database table. These values were date/time, flight data file name, max power, max power units, distance output, and distance output units (Figure 22).

The lowest level is where the execution of models occurs. This was modeled at the terminal level. This is the level where the stakeholders not only execute the analysis but also provide inputs to the systems that are part of the Digital Thread. Figure 23 illustrates the data flowing through ModelCenter, ModelCenter MBSE, MagicDraw, Node.js and MATLAB. At this level, all three data interfaces are utilized with ModelCenter and ModelCenter MBSE acting as the conductor.
6. Visualizing Results

Visualizing data collected through sensors or the outcomes of analyses is key. To that end the SQL database containing structured data was connected to Tableau and the data was used to generate plots of interest (Figure 24).

IV. Results

As mentioned previously, the Digital Thread supports the verification, calibration and validation of models through the seamless integration of models and data. In the context of this research, data had to be generated for importation into the Digital Thread. The physical aircraft completed a mission and a wide range of data was collected into the aircraft’s flight controller. The aircraft data was then extracted, file format converted, and data cleaned by keeping and processing only the relevant data. The relevant data variables were: Velocity, Flight Time, Energy Used, Voltage, Battery, Amp-Draw, and Throttle Input. The files containing the useful aircraft raw data were manually uploaded to a GitHub repository. These GitHub files were cloned to a location on the local drive, which ModelCenter can access. ModelCenter accessed the cloned files and executed the battery model and Write-to-SQL script. The Write-to-SQL function was created in ModelCenter to automatically collect the battery analysis metadata and write them to the SQL database. For emphasis, the battery model needs to be manually executed in ModelCenter, as opposed to a situation where the model is automatically executed when new cloned files are available.

ModelCenter then executed the battery model and generated some output data and plots, such as the one captured in Figure 25 where the energy consumed is plotted against the distance travelled. This figure shows agreement between the simulation (predicted) and actual (real) data (i.e. the aircraft travelled similar distances for the same energy consumed), indicating that the model is valid.

In addition to the plot generation and recommendations for the propulsion system, ModelCenter automatically verified the power draw and distance travelled output values against the propulsion system requirements. Figure 26 shows that the aircraft’s propulsion system for the mission under analysis indeed satisfied the two requirements, namely battery range and power rating.

![Digital Backbone Internal Block Diagram](image)

**Fig. 24** Digital Backbone Internal Block Diagram

![Output from the battery model showing the energy versus distance for both the real aircraft and the predicted values](image)

**Fig. 25** Output from the battery model showing the energy versus distance for both the real aircraft and the predicted values
After ModelCenter completed the analysis, it automatically uploaded the battery model analysis metadata to the SQL database. This data can be extracted for further analysis and can be imported into Neo4j to update the necessary models/nodes. Finally, since new functional data files were generated in the execution process, nodes representing these files were created in Neo4j and their relationships, locations, and attributes captured accordingly.

Fig. 26 A representation of the extraction of data and importing it into the Thread for Execution

V. Conclusion

The goal of this research was to develop an approach to the realization of a proof of concept Digital Thread for a minimum viable product. This was accomplished following the DoD HLA infrastructure standard and OOSEM. The Digital Thread was broken down into separate building blocks and the various blocks were implemented with selected tools (Figure 27).

The proof of concept Digital Thread created enabled bi-directionality of data, where information was transferred between a battery model, an MBSE environment, repositories and an integration platform. The Digital Thread was shown to support the verification (requirement traceability) of a physical aircraft’s propulsion system requirements, and the calibration and validation of a battery model representing that on the aircraft. Further, lifecycle elements or objects of the physical aircraft (models, functional data, processes, stakeholders, etc.) could be visualized and queried by means of a graph database.

A. Summary of Tools Used

For the creation and implementation of the proof of concept Digital Thread, six tools were used. These tools are outlined in Table 1 with their expected capability, the extent to which they fulfilled their expected needs, and some IT requirements or issues related to their usage are briefly highlighted.
B. Avenues for Future Work

Moving forward, the Digital Thread architecture created as part of this effort can be enhanced further. For example, in order to streamline the transfer of data from the minimum viable product, real-time data extraction via a transmitter on the airplane can be implemented. This would also make possible the automation of data transfer to the SQL server, rather than requiring a manual input from a user. To more completely implement the Digital Thread, all data and models should be stored in the Cloud, as envisioned, and the remaining sub-component models should be integrated in addition to the battery model. Sensitivity analyses should be performed to determine which models and parameters need to be properly calibrated. Finally, a dashboard could be created that measures the health of the Digital Thread, in addition to
other measurements of completion.

References


