5-2023

Developing Executable Digital Models with Model-Based Systems Engineering – An Unmanned Aerial Vehicle Surveillance Scenario Example

Viviana Guadalupe Lopez
The University of Texas Rio Grande Valley

DEVELOPING EXECUTABLE DIGITAL MODELS WITH MODEL-BASED SYSTEMS ENGINEERING – AN UNMANNED AERIAL VEHICLE SURVEILLANCE SCENARIO EXAMPLE

A Thesis

by

VIVIANA GUADALUPE LOPEZ

Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE IN ENGINEERING

Major Subject: Manufacturing Engineering

The University of Texas Rio Grande Valley

May 2023

DEVELOPING EXECUTABLE DIGITAL MODELS WITH MODEL-BASED SYSTEMS ENGINEERING – AN UNMANNED AERIAL VEHICLE SURVEILLANCE SCENARIO EXAMPLE

A Thesis
by
VIVIANA GUADALUPE LOPEZ

COMMITTEE MEMBERS

Satya Aditya Akundi
Chair of Committee

Erik Chumacero
Committee Member

Douglas Timmer
Committee Member

Hiram Moya
Committee Member

May 2023

ABSTRACT

Lopez, Viviana G., Developing Executable Digital Models with Model-Based Systems Engineering – An Unmanned Aerial Vehicle Surveillance Scenario Example. Master of Science Engineering (MSE), May, 2023, 88 pp., 3 tables, 33 figures, references, 58 titles.

There is an increase in complexity in modern systems that causes inconsistencies in the iterative exchange loops of the system design process and in turn, demands greater quality of system organization and optimization techniques. A recent transition from document-centric systems engineering to Model-Based Systems Engineering (MBSE) is being documented in literature from various industries to address these issues. This study aims to investigate how MBSE can be used as a starting point in developing digital twins (DT). Specifically, the adoption of MBSE for realizing DT has been investigated, resulting in various literature reviews that indicate the most prevalent methodologies and tools used to enhance and validate existing and future systems. An MBSE-enabled template for virtual model development was executed for the creation of executable models, which can serve as a research testbed for DT and system and system-of-systems optimization. This study explores the feasibility of this MBSE-enabled template by creating and simulating a surveillance system that monitors and reports on the health status and performance of an armored fighting vehicle via an Unmanned Aerial Vehicle (UAV). The objective of this template is to demonstrate how executable SysML diagrams are used to establish a collaborative working environment between multiple platforms to better convey system behavior, modifications, and analytics for various system stakeholders.

DEDICATION

This thesis is dedicated to my loving parents, Rolando and Zulema Lopez, your unwavering love and support have been the driving force behind my pursuit of excellence. Dad, thank you for demonstrating what it means to work diligently. This accomplishment was driven by the sacrifices you made. I hope to continue to make you proud. Mom, you have equipped me with the resilience and insight to tackle any challenge that may arise. You not only encouraged me to pursue a career in engineering but to always aspire for greater things, and for that, I am infinitely grateful. My success is our success.

To my mentor, Dr. Satya Aditya Akundi, your guidance, encouragement, and wisdom have been invaluable to me throughout this journey. Your dedication to excellence has challenged me to push past my limits and become the best version of myself. I am fortunate to have had you as my mentor and grateful for the impact you have had on my life. I am hoping that this is only the beginning of our collaboration.

To my loving husband, Colton Gonzales, your steadfast support, and compassion have been a constant source of comfort and strength. You have been my solid ground, and I can't imagine making it through this journey without you. All of this work is for us. I love you with all my heart, and I cannot wait to keep building our future together.

This thesis is dedicated to you all, and I hope it serves as a testament to your unwavering love and support.

ACKNOWLEDGMENTS

This work was supported by The University of Texas Rio Grande Valley (UTRGV) Presidential Research Fellowship (PRF) Award. The author wishes to express sincere gratitude for their financial support. The author would also like to express her appreciation for the open-source resources provided by Chun-Wei, Kong's “6-Dof Quadcopter Simulation and Control” project, and Saulius Pavalkis' “Aircraft Radar Display SysML MagicGrid Sample with Simulation and Analysis” tutorial. I'm hoping that by building on these foundations, I've produced useful new insights into the world of Model-based Systems Engineering.

The author would also like to thank Dr. Satya Aditya Akundi, Dr. Erik Chumacero, Dr. Douglas Timmer, and Dr. Hiram Moya for their expertise and assistance in completing this thesis.

TABLE OF CONTENTS

Page

ABSTRACT iii
DEDICATION iv
ACKNOWLEDGMENTS v
TABLE OF CONTENTS vi
LIST OF TABLES viii
LIST OF FIGURES ix
CHAPTER I. INTRODUCTION 1
Systems Engineering 2
Model-Based Systems Engineering 4
Modeling Tools 5
Modeling Languages 7
Systems Modeling Language 9
Digital Twins 10
CHAPTER II. INTERSECTION OF MODEL-BASED SYSTEMS ENGINEERING AND DIGITAL TWINS 15
Application of MBSE In the Context of Digital Twins 15
A Brief Review on the Use of MBSE for Digital Twin Development 17
Aerospace 17
Defense 18
Healthcare and Medical Industries 19
Manufacturing 20
A Brief Review of the Use of MBSE Tools & Languages 21
Benefits and Challenges of MBSE Utilization for Digital Twin Development 24
Benefits 25
Challenges 26
Identified Research Gap and Contributions 28

CHAPTER III. TEMPLATE USED FOR CREATING AN EXECUTABLE DIGITAL MODEL USING MODEL-BASED SYSTEMS ENGINEERING....29
Understanding the Differences between Digital Model, Digital Shadow, and Digital Twin 29
MBSE-Enabled Template for Varying Virtual Models....32
CHAPTER IV. TEMPLATE APPLICATION – A CASE STUDY ON THE APPLICATION OF MBSE FOR A UAV SURVEILLANCE SCENARIO....38
Scenario Based Testing....38
Operational Scenario....40
Conceptual Scenario....44
Problem Domain....44
Solution Domain....49
Executable Scenario....51
SoS-A Simulation....51
Shared Workspace....53
Computational Platform - MATLAB....54
UAV 6DOF Dynamics....57
Visualization....63
Getting Started with Unity....63
Unity Capabilities....68
CHAPTER V. DISCUSSION....73
Limitations and Gaps in Related Work....73
Data Sources....73
Executable Models....73
Virtual Model Types....75
Challenges and Lessons Learned....75
CHAPTER VI. CONCLUSION....77
Future Work....78
REFERENCES....80
APPENDIX....85
BIOGRAPHICAL SKETCH....88

LIST OF TABLES

Page

Table I: Tools Utilized for MBSE Approaches 22
Table II: Virtual Model Types and The Corresponding SysML Model Diagram(s) Required 34
Table III: Mission Communication Requirements 45

LIST OF FIGURES

Page

Figure 1: Data Flow from Physical System to Virtual Model Type 11
Figure 2: A Template for Developing a Digital Model Using MBSE 30
Figure 3: A Template for Developing a Digital Shadow Using MBSE 31
Figure 4: A Template for Developing a Digital Twin Using MBSE 32
Figure 5: Developing Varying Virtual Models from a Physical System using an MBSE-Enabled Template 33
Figure 6: Breakdown of Surveillance Scenario 39
Figure 7: Surveillance Scenario 42
Figure 8: Lost Connection between UAV and GCU 42
Figure 9: Functions and MoEs to Stakeholder Needs 45
Figure 10: Operator Use Case 47
Figure 11: BDD for System Port Connection 48
Figure 12: GCU Operation Mode Activity Diagram 48
Figure 13: GCU Display Screen Activity Diagram 49
Figure 14: System Requirements 50
Figure 15: IBD Mission Communication Duration Times 51
Figure 16: GCU Operator Screen with Imaging Data 52
Figure 17: Block Definition Diagram, Defining MATLAB Code 55
Figure 18: Simulating Test Block 56
Figure 19: Updated Test Parameter 56
Figure 20: MATLAB-Simulink Simulation Activity Diagram 60
Figure 21: Simulating Activity Diagram and Establishing Shared Workspace 61
Figure 22: UAV Flight Path Simulation 62
Figure 23: C++ Code for Client Set Up 64

Figure 24: C# Script For Establishing Unity as A Server.....65
Figure 25: Successful Connection Between MATLAB and Unity 66
Figure 26: UPD Send Block 66
Figure 27: Simulink Model for MATLAB-Unity Shared Workspace 67
Figure 28: C# Script for UAV Target Positioning.....67
Figure 29: SOMUA_S35 Model Tank in Unity World 68
Figure 30: UAV Target Scenario 69
Figure 31: C# Script for UAV Camera.....70
Figure 32: C# Script for UAV Body.....70
Figure 33: UAV Camera Settings.....71

CHAPTER I

INTRODUCTION

The demand for greater flexibility and competitiveness in today's manufacturing and production sectors has resulted in an increase in complexity that is causing inconsistencies in the iterative exchange loops of the system design process. To address such issues and complexities, there is a growing industry movement for organizations to migrate from document-centric concepts and applications to model-centric principles and solutions. In this chapter, an introduction to systems engineering (SE), model-based systems engineering (MBSE), and digital twins (DT) is explored on how these concepts and approaches are being applied to complex systems and system of systems (SoS). Complex systems are the center of systems engineering, a field that addresses their design, construction, and maintenance. Aircraft systems, defense systems, and production facilities can all benefit from a methodical approach to organizing, building, and maintaining a system (Liu, 2021); (Lee, 2021); (Glatt, 2021). In model-based systems engineering, models are used to describe a system's components, activities, and interconnections. By simulating, testing, and optimizing the system in advance of development, MBSE helps engineers handle the intricacy of large-scale systems. Digital twins are virtual representations of physical systems that emulate the physical system's behavior and properties (Grieves, 2017). DTs utilize data from sensors, simulations, and other sources to provide a real-time view of the system's performance, thereby enabling engineers to monitor and optimize the system in real-time. There has been a coordinated attempt, across literature, to use SE, MBSE,

and DT for complex systems and systems of systems for improved design, operation, and maintenance (Grieves, 2017). In the aerospace industry, for instance, MBSE and DTs are being used to improve the efficiency and safety of aviation systems before development even begins (Bachelor, 2019); (Li et al., 2019). The industrial sector is making use of SE, MBSE, and DTs to enhance the effectiveness of their operations (Liu et al., 2021); (Tschimer, 2015). Engineers can test and model production processes using MBSE and DTs to foresee and eliminate delays or errors, thus maximizing output and minimizing costs. For developing and managing complex systems and systems of systems, SE, MBSE, and DTs are all extremely useful approaches. Engineers can optimize the performance, safety, and effectiveness of these systems by employing these methods. The first chapter examines the various terms, definitions, tools, and languages pertinent to MBSE and Digital Twins.

Systems Engineering

Systems engineering is a methodical and comprehensive strategy for addressing system complications, taking into account every stage of the system's life cycle from design and development to retirement. INCOSE defines systems engineering as "…*a transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods*" (Sillitto et al., 2019). Defining system requirements, evaluating and improving system performance, identifying and reducing risks, and ensuring the system is reliable, manageable, and viable are all part of this approach. The advancement of powerful military technologies like radar and missile defense during the middle of the 20th century is considered to be the starting point of systems engineering (Sage & Rouse, 2014). The practice of planning and executing the

development of such complex systems was given the name "systems engineering" around the 1950s.

In *Principles of Complex Systems for Systems Engineering*, Sheard and Mostashari define complex systems as "*systems that do not have a centralizing authority and are not designed from a known specification, but instead involve disparate stakeholders creating systems that are functional for other purposes*…" (Sheard & Mostashari, 2009) There are complications that may develop when system stakeholders are unfamiliar with all of the technological aspects of a system. Primarily, it can make it challenging to communicate technical information to non-technical stakeholders, which can lead to misunderstandings or misinterpretations of system information or performance. This can lead to overblown expectations, confusion, or premature decisions. Non-technical stakeholders may also not completely comprehend the trade-offs and limitations that exist within a system. They may request modifications or improvements that are not technically feasible or that compromise the system's overall performance or dependability. These stakeholders may have different priorities and objectives than the engineers who are developing the system. This can result in conflicts and disagreements regarding the project's direction, which can cause delays, budget overruns, or even project failure. Overall, effective communication and collaboration between technical and non-technical stakeholders is essential for ensuring the success of a complex system. This requires clear and concise communication of technical information, a shared understanding of goals and priorities, and a willingness to work collaboratively to navigate obstacles, which systems engineering begins to address (Sillitto et al., 2019); (Henderson, 2021); (Bretz et al., 2016).

Across literature, Systems Engineering has been used in the development of complex systems with the aim of enhancing system performance (Hause, 2019). As previously mentioned,

a Systems Engineering approach provides a structured method for system design and development that takes into account all aspects of a system, from its conception to its eventual retirement or disposal (Sillitto et al., 2019). This approach can enhance system performance, dependability, and efficiency. Systems Engineering also provides a comprehensive approach to risk management that includes identifying, analyzing, and mitigating risks throughout the life cycle of a system. As a result, there will be fewer system failures and lowered costs. Throughout the life cycle of a system, systems engineering necessitates the collaboration of multiple disciplines and stakeholders (Bajaj et al., 2011). This approach can lead to enhanced communication and comprehension, resulting in more informed decisions and more efficient problem-solving. The primary goal is to deliver a high-quality solution that satisfies the customer's and stakeholders' requirements (Sage & Rouse, 2014).

Model-Based Systems Engineering

As defined in the 2007 INCOSE Model-Based Systems Engineering Initiative, Model-based Systems Engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification, and validation, beginning in the conceptual design phase and continuing throughout development and later life cycle phases (Friedenthal et al., 2007). The adoption of MBSE has proven successful in various industries by providing a clear and comprehensive system model that can be examined for stability and reliability (Phanden et al., 2021). Several studies have demonstrated that using MBSE techniques can enhance interconnectivity among system stakeholders (Bretz et al., 2016). By creating a comprehensive system model that can be viewed from multiple perspectives, alternative solutions can be assessed, and their implications understood. This improved understanding of the system and its architecture leads to increased dependability. Identifying potential issues before they arise allows

for timely corrective actions, reducing the risk of system failures. As a result, MBSE is increasingly being embraced by organizations seeking to optimize their systems engineering processes and enhance the quality of their products or services.

Three key components are the foundation for Model-Based Systems Engineering (MBSE): a modeling technique, tool, and language (Delligatti, 2013). A modeling technique is a collection of procedures and rules to build a system model in a virtual environment. This involves the procedures and techniques for generating system requirements, designing the system, analyzing its performance, verifying and validating it, and managing the system's data. A well-defined modeling technique is essential for achieving consistency in system interoperability and model development.

Modeling Tools

A modeling tool is a software application that aids system model creation, analysis, and visualization (Delligatti, 2013). These tools are meant to adhere to various modeling language standards, enabling the production of reliable models in the respective modeling language. Modeling tools come in various formats, including independent desktop programs and browser-based web apps. They offer a graphical user interface that enables users to construct and edit models utilizing symbols, diagrams, and other visual representations.

Magic System of Systems Architect (SoS-A) is a No Magic, Inc. developed software application used for designing and analyzing complex systems of systems (“Magic Systems of Systems Architect Documentation,” n.d.). The term "system of systems" (SoS) refers to a network of interconnected systems that perform a specific task. SoS-A supports multiple modeling languages and notations, including Systems Modeling Language (SysML), Unified

Modeling Language (UML), and others, allowing users to describe and test a wide range of system-of-systems configurations. The program provides a full setting for building and evaluating intricate SoS systems, including the means to specify system interfaces, handle relationships, and identify issues before they occur. SoS-A's ability to integrate with other modeling and simulation tools, such as MATLAB and Simulink, enables users to construct a more comprehensive view of their SoS. Additionally, the software includes collaboration tools, allowing multiple users to simultaneously work on the same SoS model. Magic System of Systems Architect has been designed to assist organizations in designing, simulating, and managing complex systems of systems by providing an all-inclusive environment for modeling, analyzing, and optimizing SoS architectures ("Magic Systems of Systems Architect Documentation," n.d.).

IBM Rational Rhapsody, also known as IBM Rhapsody, is another software development tool used for modeling, designing, and implementing real-time and embedded systems ("Engineering systems design rhapsody – overview," n.d.). It is developed and maintained by International Business Machines Corporation (IBM). Rhapsody uses UML and SysML to create graphical models of systems and processes. These models can include various types of diagrams, such as class diagrams, sequence diagrams, and activity diagrams, among others. The software provides a comprehensive environment for designing, testing, and deploying software applications, including the ability to generate code in various programming languages, such as C++, Java, and Ada. It also includes features for testing and debugging software, as well as for managing version control and project collaboration. IBM Rhapsody is designed to help organizations improve the efficiency and quality of their software development processes by

providing a comprehensive modeling and development environment for creating complex real-time and embedded systems ("Engineering systems design rhapsody – overview," n.d.).

Organizations can also benefit from using Capella, a free and open-source modeling and simulation program that was developed by the PolarSys Industry Working Group of the Eclipse Foundation ("Features and benefits," n.d.). It is intended to assist engineers and developers in creating complex systems and models using the graphical modeling language and approach Arcadia, a specific implementation of the OMG SysML standard. Block diagrams, sequence diagrams, and state diagrams are just some of the diagram types that can be created with Capella. It also features tools for simulation and verification, allowing users to test and analyze the behavior of their models/systems. Capella's support for joint development is a crucial element that enables numerous people to work on the same model concurrently. In order to guarantee that models are well-documented and in accordance with project objectives and specifications, the software also includes features for traceability and monitoring. Capella's primary goal is to aid organizations in enhancing their systems engineering procedures by giving them an all-encompassing setting in which to model and simulate complicated systems and models ("Features and benefits," n.d.). It is particularly useful for systems that are safety-critical or that have intricate component interactions and communications.

Modeling Languages

A modeling language is a formalized language that describes the rules and standards for building system models (Delligatti, 2013). It provides a standardized language and syntax for communicating system requirements, design specifications, and other system-related information. Modeling languages may be graphical, like the Unified Modeling Language (UML), Systems Modeling Language (SysML), and Arcadia, or textual, like Architecture Analysis and

Design Language (AADL) (Feiler et al., 2006). Since each modeling language has its own rules and conventions, selecting the appropriate language for the modeled system is essential.

UML is a standardized visual modeling language used in software engineering to represent software systems and processes ("What is UML," n.d.). SysML is a specialized version of UML designed for modeling complex systems and processes ("ABOUT THE OMG SYSTEM MODELING LANGUAGE SPECIFICATION VERSION 1.7 BETA," n.d.). It extends UML with additional constructs and notations that are specific to MBSE. A more extensive definition of SysML will be explored later on. Arcadia is a systems engineering method that uses the Arcadia language to describe and analyze complex systems. Arcadia is based on SysML and extends it with additional notations and constructs, making it more suited to various specific domains ("Let yourself be guided with Arcadia" n.d.). AADL (Architecture Analysis and Design Language) is a modeling language designed for describing the architecture of real-time and embedded systems ("Architecture analysis and Design Language (Aadl)," n.d.). It is used to represent the structure and behavior of the system at various levels of abstraction and provides a way to formally analyze the system's properties, such as timing and resource utilization. These modeling languages and methods are used to represent complex systems and processes, helping engineers and developers to better understand, analyze, and design software and systems. They are only some of the widely used modeling languages in industry and academia and are essential resources for software engineering, systems engineering, and MBSE practitioners and professionals.

In this thesis, the Magic Systems of Systems Architect (SoS-A), a modeling tool that actively supports system development architectural frameworks, is used. The software allows for the execution of SysML models through the Magic Model Analyst, an execution framework

plugin (“MagicDraw - CATIA - Dassault Systèmes®,” n.d.). There are several modeling languages available, each with its strengths and weaknesses. SysML is chosen as the primary modeling language in this study due to its more flexible and all-encompassing semantics. It is well-suited for expressing performance and quantitative metrics. Through the use of SysML diagrams, the complex architecture of a system can be better understood and communicated among stakeholders.

Systems Modeling Language

As priorly mentioned, SysML stands for Systems Modeling Language, a graphical modeling language used for designing, analyzing, and specifying complex systems. SysML provides a set of notations and diagrams to enable a more comprehensive and organized representation of system architecture. There are nine diagram types in SysML, each with a specific purpose (Delligatti, 2013):

1. Block Definition Diagrams (BDDs) - displays blocks and value types and their relationships, such as system hierarchy and classification trees.

2. Internal Block Diagrams (IBDs) - indicate the internal structure of a single block by showing the connections between the internal parts of a block and the interfaces between them.

3. Use Case Diagrams - to model the interaction between the system and its users or external systems.

4. Activity Diagrams - specify a behavior, focusing on the flow of control and the transformation of inputs into outputs through a sequence of actions.

5. Sequence Diagrams - used to model the behavior of a system over time, depicting the interactions between different system components or actors.

6. State Machine Diagrams - Identify a behavior by showing the set of states of a block and the possible transitions between those states in response to event occurrences.

7. Parametric Diagrams - expresses how constraints are bound to the properties of a system, supporting engineering analyses and trade studies of candidate physical architectures.

8. Package Diagrams - used to organize the system models into groups or packages, allowing for more straightforward navigation and management.

9. Requirements diagrams - display text-based requirements and the relationships between requirements and the other model elements that satisfy, verify, and refine them.

Each diagram type in SysML plays a unique role in modeling the architecture and behavior of a system, and they can be used together to create a comprehensive model of a complex system (Delligatti, 2013). SysML is the primary language that is utilized throughout this research work.

Digital Twins

A digital twin (DT) is an interactive, real-time digital representation of a system or service utilizing onboard sensor data and Internet of Things technology. Data from the physical system is used to develop and enhance the digital twin by providing an accurate and consistent, real-time model of a physical system. The concept of a digital twin, initially introduced by Michael Grieves in 2002, is gaining traction in the MBSE community (Grieves, 2017). A digital twin is continuously updated with the corresponding physical system and performance data

throughout its system life cycle (Kritzinger et al., 2018). However, a review of scientific articles proved that a precise definition of a DT has yet to be developed as definitions vary across different domains. According to Kritzinger et al., there are three virtual representation levels of a digital twin. Each level has a distinct purpose and scope throughout the system's lifecycle, helping with decision-making and addressing challenges. Depending on the level of data integration, some virtual models are created manually and have no physical data from the product/systems, while others are extensively interconnected with real-time data exchange (Kritzinger et al., 2018). It is observed that the terms digital model (DM), digital shadow (DS), and digital twin (DT) are used interchangeably across literature based on the level of interoperability among a virtual model created and its corresponding physical system. Figure 1 attempts to illustrate the core differences between a DM, DS, and DT (Lopez & Akundi, 2022).

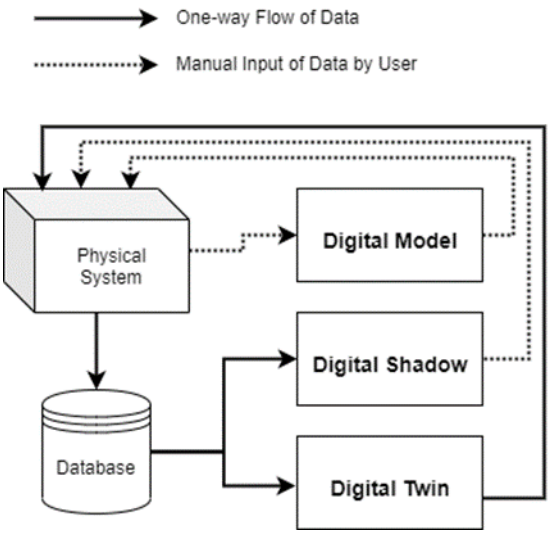

Figure 1. Data Flow from Physical System to Virtual Model Type

A DM is a digital depiction of a physical system that does not utilize any computerized data exchange between the physical system and the virtual model (Kritzinger et al., 2018). Data from the physical system is manually input, negating the real-time exchange of data between the

physical system and DM. The level of complexity can only pertain to the detail of physical system components and environment. Any information gained from a DM will not directly affect the physical system. As seen in Figure 1 information about the state of the physical system is manually input by a user to the digital model (Lopez & Akundi, 2022). This manual exchange of information is represented in the figure as a dotted line.

A DS is all that a DM is with an addition of an integrated one-way data flow between the state of an existing physical system and the state of a virtual model (Kritzinger et al., 2018). Any modification made to the physical system will result in an automated update to the DS, which is accomplished via an information exchange that is processed by a database. This automatic one-way exchange of information is represented as a solid line in Figure 1 (Lopez & Akundi, 2022). However, a change in the virtual model will not directly affect a change in the physical system. Changes determined by the DS must be manually implemented in the physical system by the user.

A DT has real-time interconnectivity between an existing physical system and the virtual model (Kritzinger et al., 2018). Changes in the virtual model can directly affect the physical system. The DT can also make decisions that change the performance, functionality, or status of the physical system. Other physical elements, such as the environment of a physical system, may affect the status of the DT as information is automatically transmitted through a database. A change in the physical system causes a change in the DT state and vice versa as represented by Figure 1.

Digital Twins (DT) have had a significant impact on the design and optimization of physical systems (Delbrügger & Rossmann, 2019). The three pillars of Model-Based Systems Engineering (MBSE) can and have been used to initiate DT development. MBSE techniques can

be used to streamline workflows, resulting in effective system development. Throughout each phase of a system's lifecycle, MBSE users can use modeling and simulation data to create a Digital Twin of the physical system(s) (Liu et al., 2021). Depending on the functional, operational, and other system requirements, the method used to generate the DT via MBSE may vary. The outcome is a DT that accurately reflects the functionality and behavior of the system. MBSE can aid in establishing synchronization between structural, technical, inspection, software, and other engineering disciplines and elements of a physical system.

Over the years, the analysis of system requirements, behavior, structure, and parameters, as well as their representation in a modeling language such as SysML, were used to construct a DT using MBSE (Lopez & Akundi, 2022). The integration of MBSE makes the creation of DTs organized and more efficient. The process of developing a DT can be divided into multiple phases using MBSE (Pang et al., 2021). These phases contribute to the systematic development of the DT and guarantee that the functionality and behavior of the DT consistently reflect the physical system. Thus, engineers can gain accurate insights into the performance and behavior of a physical system throughout its lifecycle (Bajaj et al., 2011). The implementation of MBSE techniques in the construction of DTs has played a crucial role in attaining synchronization across various engineering disciplines. Using MBSE techniques facilitates the identification and resolution of system complications that may arise during the physical system’s development. This strategy has also led to the development of efficient and dependable optimized systems. By employing MBSE techniques and tools, engineers are able to construct DTs that aid in the simulation of various scenarios, resulting in the creation of efficient solutions that meet system and stakeholder requirements (Schluse et al., 2017).

The sections and chapters that follow are expanded versions of previously published articles by the author for this research. Through a literature review, Chapter II investigates the intersection of MBSE and Digital Twins. Following is a summary of the observed challenges and benefits found in the literature regarding the use of MBSE for the development of digital twins. The development of an MBSE-enabled template for creating varying virtual models and how it can be used to generate executable models is described in Chapter III. Chapter IV then goes on to describe how this template was used in a case study of MBSE for UAV surveillance. This chapter is divided into three major sections that correspond to the operational, conceptual, and executable scenarios of the UAV surveillance scenario. Then, in Chapter V, the most pertinent observed gaps and limitations in both the literature and this research will be discussed. Finally, Chapter VI will include concluding thoughts and future work.

CHAPTER II

INTERSECTION OF MODEL-BASED SYSTEMS ENGINEERING AND DIGITAL TWINS

A digital twin (DT) is an interactive, real-time digital representation of a system or a service utilizing onboard sensor data and Internet of Things (IoT) technology to gain a better insight into the physical world (Lopez & Akundi, 2022). With the increasing complexity of systems and products across many sectors, there is an increasing demand for complex systems optimization. Digital twins vary in complexity and are used for managing the performance, health, and status of a physical system by virtualizing it. The creation of digital twins enabled by Model-based Systems Engineering (MBSE) has aided in increasing system interconnectivity and simplifying the system optimization process. More specifically, the combination of MBSE languages, tools, and methods has served as a starting point in developing digital twins (Lopez & Akundi, 2022). This chapter discusses how MBSE has previously facilitated and used in the development of digital twins across various domains, emphasizing both the benefits and disadvantages of adopting an MBSE-enabled digital twin creation.

Application of MBSE In the Context of Digital Twins

The information and data acquired through the use of digital twins have significantly influenced the design and optimization of physical systems. Current research shows that the three pillars of MBSE have been employed as a foundation for developing digital twins. By employing MBSE techniques, processes can be streamlined for greater efficiency. MBSE languages,

methods, and tools such as SysML, Cameo Systems Modeler, and MagicGrid have been utilized to develop system models gradually (Liu et al., 2021). SysML has been previously translated into a programming language like Java and utilized to mimic the system model in a simulation engine (Liu et al., 2021). Cameo Systems Modeler is a cross-platform MBSE environment that allows users to create, track, and digitize system characteristics using SysML model diagrams (Liu et al., 2021); (Wang et al., 2021); (Tsui et al., 2018). System stakeholders and contributors can then easily track system models and those models are then saved as XMI files, or distributed to documents, graphics, and web interfaces. MagicGrid enables separating the process of creating a system model into three domains: problem, solution, and implementation (Liu et al., 2021); (Wang et al., 2021); (Tsui et al., 2018). Comparing simulation outputs to actual results can reveal vital information about the physical system's performance, health, and status. Engineers can create event-driven or agent-based simulations to investigate the behavior and interactions of the DT using an appropriate MBSE tool and language (Madni et al., 2019). MBSE provides a standard guideline for system management, system-to-system architecture, and operational scenarios to promote synchronous model creation and enhance the re-usability of model data. With MBSE, users can gather model data from various engineering and manufacturing products and processes. Users of MBSE have utilized modeling and simulation data to generate a DT of a physical system at each stage of its correspondent lifecycle phase (Pang et al., 2021).

The MBSE approach used to create a digital twin is dependent on functional, operational, and other system requirements. This method results in a digital twin that accurately reflects the system's behavior and functionality. Previous instances of using MBSE to generate a digital twin involved analyzing a system's requirements, behavior, structure, and parameters and representing them using a modeling language like SysML. Integrating MBSE aids in the establishment of

synchronization across different engineering disciplines such as structural, technical, inspection, software, and other various elements of a physical system (Phanden et al., 2021).

A Brief Review on the Use of MBSE for Digital Twin Development

A thorough review of literature revealed that employing MBSE in developing digital twins has numerous benefits, such as enhancing system comprehension, improving system efficiency, and reducing development costs. MBSE can provide a comprehensive view of the system under development, allowing for an improved understanding of the system's behavior, requirements, and limitations. This information can be leveraged to optimize the system's design, reducing development time and costs. However, the review also highlighted several practical issues that must be addressed to successfully employ MBSE in digital twin creation. One such issue is the requirement for specific knowledge and skills, including expertise in systems engineering, modeling, and simulation. To create an effective digital twin, the user must have a thorough understanding of the system's behavior, structure, and operation, as well as the relevant modeling techniques and tools. Another significant issue is the complexity of integrating data from numerous sources, which can result in a lack of consistency and accuracy in the digital twin model. Additionally, the need for data validation and verification can further complicate the process. Despite these challenges, the benefits of employing MBSE in digital twin creation are significant, and efforts are being made to address these issues through the development of more advanced modeling techniques and tools.

Aerospace

By allowing for real-time monitoring and control, Digital Twin technology has enhanced quality control throughout the machining process for aerospace component production (Liu et al.,

2021). The machining process is modeled using a biomimicry-based technique. The resulting multi-physics digital twin model includes a model display module, a data monitoring module, and a process display module (Liu et al., 2021). An adaptive digital twin model can be built for the entire product life cycle, starting from the planning of the manufacturing process and continuing with real-time monitoring of the machining state.

Biological mimicry is a phenomenon in which one species has evolved to closely resemble another species to gain an advantage, such as protection from predators or access to food. This can occur through physical resemblance, as seen in the case of a harmless snake that mimics a poisonous species, or through behavioral mimicry, as when a predator imitates the sound of a distressed animal to lure its prey. Changes in geometry, behavior, and context during machining may be reflected and considered by using digital twin mimic models (DTMM) (Liu et al., 2021). Using the process path as a guide, the digital twin mimic model incorporates detailed information on the product's geometry, physics, and production procedure. The geometry, behavior, and context model are kept in the DTMM, and represented using UML. Data objects are stored in a tree structure in XML for cross-platform compatibility with the DTMM, with properties of the data objects represented in the XML tree nodes (Liu et al., 2021). Because of this, data included inside the digital twin model may be efficiently organized, managed, and optimized over its entire lifecycle.

Defense

A process for creating a digital twin of an Unmanned Aircraft System (UAS) that can provide route selection capabilities from the perspective of Mission Engineering (ME) has been developed (Lee, 2021). This case study illustrates the methodology, including employing a UAS for a Last Mile Delivery (LMD) mission and recommending an appropriate path to the user using

a route optimization module. Lee provides a framework for developing a digital twin of the UAS, which includes defining stakeholder requirements and use cases, quantifying system parameters and mathematical expressions, and organizing the data with CAMEO Enterprise Architecture software and SysML to produce a simplified architecture of the UAS for the LMD mission (Lee, 2021). Multi-Attribute Utility Theory (MAUT) is the foundation of the used optimization module, which evaluates user-determined success requirements for the UAS mission. Time to target, remaining battery power, and hazard likelihood are considered to determine the optimal route. The success rate of the LMD mission's case study can either increase or decrease based on the identified threats. To model the physics of the UAS, parametric equations are utilized. The sorted information is then sent to ModelCenter for simulation of the case study. During the simulation, ModelCenter will highlight the necessary inputs and display the computed results visually (Lee, 2021). The computed paths will assist the UAS user in making informed decisions. This MBSE approach, along with decision-support technologies, improved the efficiency of the system and its interactions with the environment.

Healthcare and Medical Industries

The advancement in creating digital twins extends beyond the optimization of individual or multiple systems. Researchers are exploring the potential of utilizing DT technology to virtualize humans and human organs. An ongoing study, funded by the Air Force Institute of Technology, demonstrated that a digital twin of a human individual could be developed using SysML to organize data and information (Pirnstill et al., 2022). Mathematical modeling was carried out using parametric diagrams, and data was organized through package diagrams. The human body was initially represented using block definition diagrams. To interpret the use cases, the DoD product development technique was utilized. The implementation of this technique

highlights the numerous ways in which MBSE techniques can be used to structure and define a system, such as the human body, in a digital environment. The objective of this study was to create a digital twin by establishing a two-way information exchange between the real world and its digital representation. The digital twin of a human can be further enhanced by utilizing sensors to provide real-time data. In a given scenario involving one or multiple individuals, eye movements, injuries, heart rate, brain electrophysiologic signals, blink rates, and timing can all be monitored to collect information that can depict or predict human attributes and actions (Pirnstill et al., 2022). Real-time visualization of a person's health status could be highly beneficial to the medical community.

Manufacturing

Cyber-physical production systems are highly versatile and adaptable, allowing for the production of individualized goods in low-quantity production runs (Glatt, 2021). However, due to the complex material fluxes within these systems, physically caused disruptions can lead to breakdowns, decreased throughput, and higher costs. One solution to this issue is to use a physics engine to model the frictional forces exerted by workpieces on material handling equipment. Researchers were able to model and simulate an experimental material handling system using UML diagrams and a Python script (Glatt, 2021). Modeling the system not only provided a visual representation but also allowed for simulation-based decision assistance, network connection, and control inputs to the physical system. The simulation environment is built using input/output data that the user loads into the digital twin. The digital twin was enhanced with diagnostic features to capture the actual material flow process, constraints, and moving components of the system. During the simulation, UML sequence diagrams depict the interactions that occur, and calculated data from the simulation can inform the user whether their

input into the existing system is optimal (Glatt, 2021). This procedure generates data that can be used for future simulations, particularly for predictions. By simulating the physics of the system, users can test and evaluate various material handling characteristics, such as the speed at which a material/component is moved or the restrictions of a mechanical load. For example, greater accelerations allow for quicker material handling but also increase the mechanical load on the carried workpiece(s). A collision detection algorithm in a physics simulation can continuously calculate the horizontal position of a workpiece with the transport mechanism, enabling either a human supervisor or the simulation itself to detect disturbances. The program records the three-dimensional coordinates of the material and the transport mechanism in every simulation frame (Glatt, 2021). With the integrated prediction function, faster material handling without sacrificing safety is now possible, which may enhance throughput and reduce disruptions.

A Brief Review of the Use of MBSE Tools & Languages

There are many different configurations in which modeling via MBSE can take place, depending on the system's demands or the stakeholders' objectives. Modeling a system may be essential in visualizing and organizing its architecture, or it may be necessary to assess and validate system behavior. Even the simulation of a physical or theoretical system is possible through modeling. There are several tools available that were either developed specifically for MBSE methods or adapted for use in MBSE, and they utilize a respective modeling language. According to INCOSE's description of MBSE, the following are examples of tools and languages previously used in literature to address system requirements, design, analysis, verification, and validation. A summary of the specified tools and languages is provided in Table I.

In any system's design and development process, system requirements are critical as they define the constraints within which a system or product must operate. Meeting or surpassing these requirements is essential for the system's success, and it ensures that the end product satisfies the demands of the stakeholders. By designing, developing, and testing a system with requirements, it becomes easier to verify that it is feasible for production, meets the user's needs, and is cost-effective. Moreover, system requirements help identify potential system challenges or limitations, allowing designers to address these issues before they become major bottlenecks in the development process.

Table I. Tools Utilized for MBSE Approaches

Lifecycle Phase	***Tool***	***Language***	***Author(s)***
System Requirements	Xtext Eclipse	DSML	Lemazurier et al.
Design	Rhapsody	UML, SysML	Sakairi et al.
Analysis	Simulink	SysML	Sakairi et al.
Verification	nuXmv	SysML	Staskal et al.
Validation	nuXmv	SysML	Staskal et al.

The development of a system necessitates a clear understanding of stakeholder needs and system requirements. Stakeholder needs are based on customer expectations, while system requirements are based on the system's design specifications (Anyanhun & Edmonson, 2018). Therefore, it is necessary to translate stakeholder needs into system requirements to influence a system's design effectively. MBSE provides methodologies to develop traceable requirements that can be accessed throughout a system's lifecycle phases. Traceable requirements ensure that

each requirement can be traced back to a specific stakeholder need, thereby establishing a clear link between the two. This link allows designers and engineers to develop a system that meets stakeholder needs while still adhering to design specifications. In addition, traceable requirements aid in managing the system's complexity by breaking down the requirements into smaller, more manageable chunks that can be implemented incrementally (Anyanhun & Edmonson, 2018). By utilizing MBSE to develop traceable requirements, organizations can ensure that the developed system meets stakeholder needs, adheres to design specifications, and is developed efficiently.

The Xtext Eclipse platform is a tool that allows developers to construct domain-specific modeling languages (DSML) and programming languages that enhance the clarity and structure of the requirements definition process (Lemazurier et al., 2017). The DSML tool enables users to describe and place a system within its context, operating modes, and transitions and communicate the expected input/output behavior in compliance with the specifications. Additionally, it enables users to specify the required operating situations. System design is complex, and it can be challenging to manage numerous components and subsystems with their own requirements, interconnections, and limitations. IBM Rhapsody, a modeling and simulation tool, can help address these factors

As mentioned in Chapter I, IBM Rhapsody enables users to design, develop, and test complex systems, such as real-time and embedded systems (Sakairi, 2013). To do so, it aids users in constructing models of a system using UML or SysML to simulate the system's behavior to identify possible issues and ensure it satisfies the specified requirements. IBM Rhapsody also offers code generation, testing, and visualization capabilities, which can save time and resources necessary to design and implement sophisticated systems. MBSE approaches often include the

integration of multiple tools/platforms. A shared workspace between Rhapsody and Simulink has previously been developed (Sakairi, 2013). Rhapsody can design a system using SysML diagrams that contain references to Simulink models, which are later used for simulation in Simulink. Verification and validation (V&V) activities ensure that the system operates as intended and may aid in identifying and resolving any faults or flaws before putting the system into operation.

nuXmv is a new symbolic model checker for synchronous finite-state and infinite-state systems analysis (Staskal et al., 2022). Specifications for successful system operations may be developed during the requirements identification phase. These needs are then organized using SysML models and translated into nuXmv. Using Linear Temporal Logic (LTL), nuXmv can determine if a system meets requirements ranging from safety-critical to high-level convenience. After importing validation criteria, nuXmv will check whether the system performs as expected (Staskal et al., 2022). MBSE provides methodologies and tools for developing traceable requirements and managing complex systems. Xtext Eclipse, Rhapsody, Simulink, and nuXmv are examples of tools that can be integrated into an MBSE approach to enhance the clarity and structure of the requirements definition process, design and develop complex systems, and verify and validate their operation.

Benefits and Challenges of MBSE Utilization for Digital Twin Development

MBSE methodologies for creating Digital Twins will vary depending on the industry and organization, as a standardized approach has yet to be established. Various modeling languages and tools are available to facilitate different forms of modeling. In the absence of a standardized MBSE approach, practitioners typically adopt specific modeling tools and languages as their preferred modeling methods. These methodologies or techniques differ across domains because

each domain has its own set of requirements and limitations for its systems. When designing an MBSE methodology, the system and its stakeholders must be taken into account. MBSE-enabled digital twin technology has the potential to provide numerous benefits for various industries. Below are some of the challenges and benefits of using and developing an MBSE-enabled digital twin.

Benefits

Considering that digital twins are digital copies of real-world objects or systems that can be used for modeling and training, MBSE practitioners can adjust model parameters and conduct experiments to improve decision support if a closed-loop modeling method is achieved (Madni, 2021). In this way, numerous use cases and scenarios can be evaluated before the actions and behaviors of the real-world components are finalized, which saves time and money. MBSE enables digital twins to be used for t visualization and simulation of not only systems and products but also people and groups. Using this approach, complicated socio-technical experiments can be conducted at a lower cost and in less time than if humans were involved (Madni, 2021).

Efficiency, collaboration, quality, maintenance, and modifications are just some of the areas where digital twins can truly excel. Designers and engineers can save time and money by fixing problems in the digital twin model through simulation and testing before the system is physically constructed. MBSE enables all parties involved in the system development process access to the digital twin and can make and view modifications in real-time, thereby fostering improved collaboration and communication. Through extensive testing and analysis of the system architecture, digital twin technology enabled by MBSE can guarantee higher-quality products. The digital twin can then be used for maintenance and modifications after the

system has been put into production, cutting down on downtime and maximizing efficiency. Since Digital Twins and the formalization of MBSE methods are both in their preliminary stages, many different industries have begun investigating how these tools can enhance their processes and capabilities. Users, stakeholders and organizations can gain insight into how their systems and processes function and where they can be enhanced by employing digital twins. Data visualization allows multiple people to better understand the system's complexity, which in turn leads to better decisions.

Digital twins enabled by MBSE are gaining popularity in various industries and for legitimate reasons. When it comes to improving productivity and output, businesses are constantly looking for new tools and techniques. Using MBSE and digital twins, systems have successfully been organized and optimized more efficiently. As the study of these areas expands, so will the associated advantages and difficulties. A more standardized approach is required for MBSE and digital twins to reach their full potential.

Challenges

Implementing MBSE-enabled digital twins requires careful consideration of several factors, including the reliability of data sources, the complexity of the model, and the accuracy of the model's representation of the real system. Addressing challenges such as data integration, validation and verification, and privacy and security will be critical for successful implementation. The validation and verification of an MBSE-enabled digital twin heavily rely on the availability of reliable and consistent data. To achieve real-time data exchange between a physical object and its digital counterpart, extensive preparation, testing, and maintenance are required. Data must be transmitted to both the modeling and simulation tool. The data used to create the digital twin must be collected from various sources and integrated into a coherent

model(s) in the respective MBSE modeling tool. Creating an accurate and comprehensive digital twin model requires considerable time and expertise, and the model can become quite complex. Ensuring the accuracy and validity of the digital twin model is crucial, and it can be challenging to verify that the model is representative of the real system. While physical tests may be more expensive overall, they offer immediate feedback. Whereas, the speed and accuracy of complicated model simulations still require further study, as simulations may encounter delays due to insufficient processing power or connectivity issues with the digital twin's physical counterpart. In addition, the term "digital twin" lacks a standardized definition, and its interpretation varies depending on the context. Some researchers have referred to their models as digital twins, even if they have not been tested or validated and do not engage in real-time data exchange. In contrast, researchers whose work involves a more sophisticated data interchange tend to avoid using the term. Given that digital twins are developed and managed across multiple platforms, models may contain sensitive data, and securing the model and controlling access to it can be challenging.

In addition to the absence of a standard definition for digital twins, there is also no formalized MBSE approach for developing digital twins. In a paper titled, "Model-Based Systems Engineering for AI-Based Systems," Sprockhoff et al. describe an AI-based threat localization system. They propose a framework for systematic development that enables the design and modeling of AI-based systems (Sprockhoff et al., 2023). Despite the fact that their work concentrates on a subject other than digital twins, the MBSE framework they develop is relevant to this thesis. The authors acknowledge the scarcity of research on the use of MBSE for developing AI-based systems, just as there is a shortage of formalized MBSE approaches for digital twin development. In chapter V, this correlation will be examined in greater detail.

Identified Research Gap and Contributions

Currently, there is no standardized MBSE methodology for developing system architectures that can be executed toward creating true digital twins. This lack of formalization also extends to the absence of continuity between different tools and platforms required to model, simulate, and visualize physical systems in a virtual environment. Real-time physical system data, system architectural modeling, model simulation, and 3D visualization are all beyond the scope of any single tool. On the other hand, there are dedicated tools that can carry out the tasks mentioned above and can be linked to one another. In the following chapter, an MBSE-enabled template for developing varying virtual models is explored. The primary goal of this template is to demonstrate how executable SysML diagrams can be utilized to generate virtual models and a collaborative workspace across several platforms.

CHAPTER III

TEMPLATE USED FOR CREATING AN EXECUTABLE DIGITAL MODEL USING MODEL-BASED SYSTEMS ENGINEERING

This section presents a mapping of an MBSE-enabled template for developing an executable model based on observations of the use of MBSE across various domains as outlined in Chapter II. It is important to first understand the key differences between a digital model, digital shadow, and digital twin to understand their context within MBSE.

Understanding the Differences between Digital Model, Digital Shadow, and Digital Twin

Digital models, digital shadows, and digital twins are all virtual representations of physical systems, each with its own unique characteristics and capabilities (Kritzinger et al., 2018). While a digital model is a simplified representation of a physical system that can be created using a modeling language such as SysML and 3D modeling software, a digital shadow incorporates real-time data from onboard sensors to provide more detailed insights into the system's behavior. A digital twin, on the other hand, is a real-time virtual replica of a physical system that is interconnected with it, allowing for the exchange of data and influencing each other.

Developing a digital model facilitated by MBSE is a cost-effective method of creating a simplified representation of a physical system that provides stakeholders with an efficient comprehension of its structure and behavior. The initial steps for developing a digital model are

to define the system's requirements, establish its design and architecture, and choose the appropriate modeling language and tool to represent it, as seen in Figure 2.

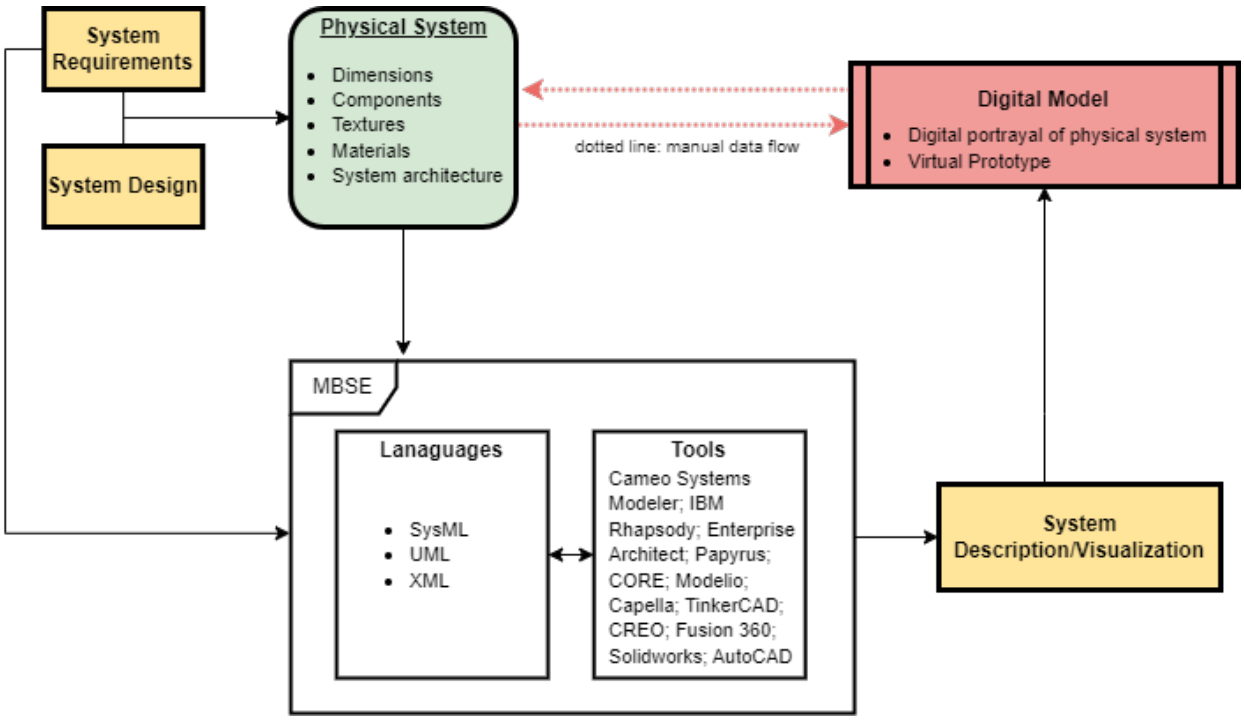

Figure 2. A Template for Developing a Digital Model Using MBSE

Developing a digital shadow requires a more comprehensive process than a digital model, including the incorporation of real-time data from onboard sensors and more dynamic modeling and simulation tools (Figure 3). The analytical information obtained from a digital shadow can provide stakeholders with invaluable insights for making informed decisions regarding the physical system.

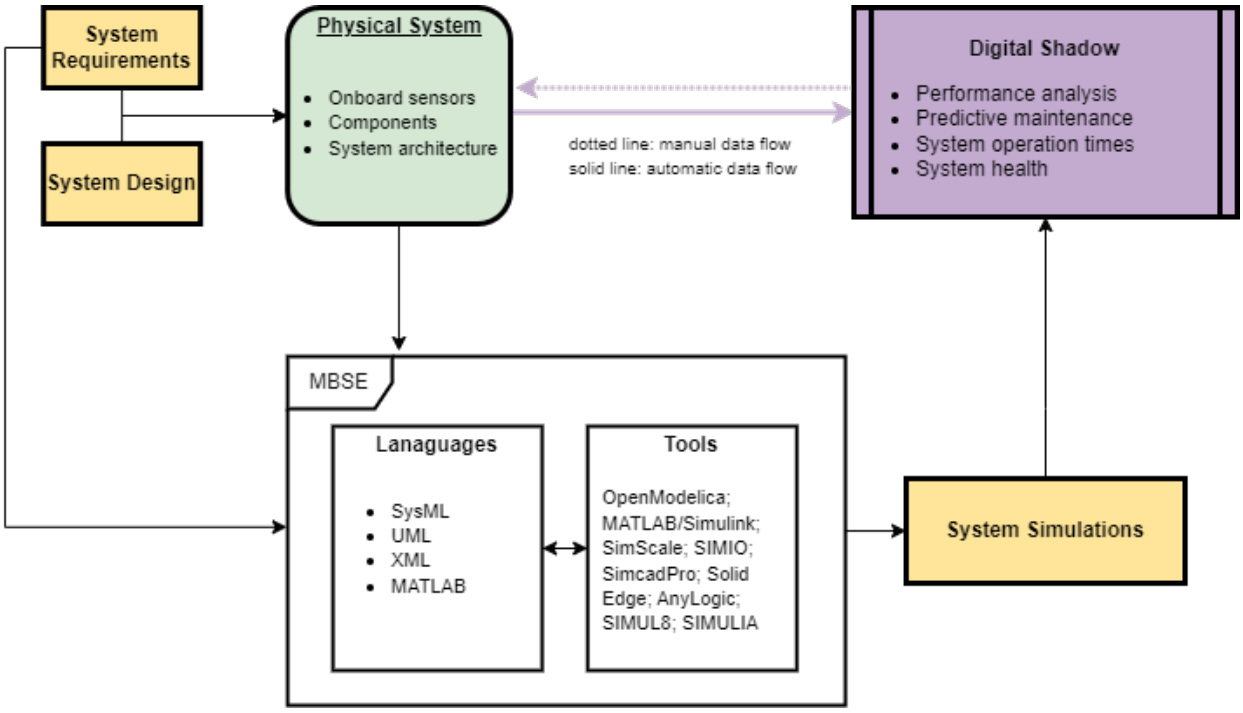

Figure 3. A Template for Developing a Digital Shadow Using MBSE

Developing a digital twin is the most complex of the three, requiring real-time data from sensors to be uploaded into the respective tool, a model library or cloud to hold all the information, and a cross-disciplinary understanding of the tools used to create it (Figure 4). It is essential to comprehend the distinctions between a digital twin, a digital shadow, and a digital model to avoid confusion and ensure that each virtual representation is used appropriately.

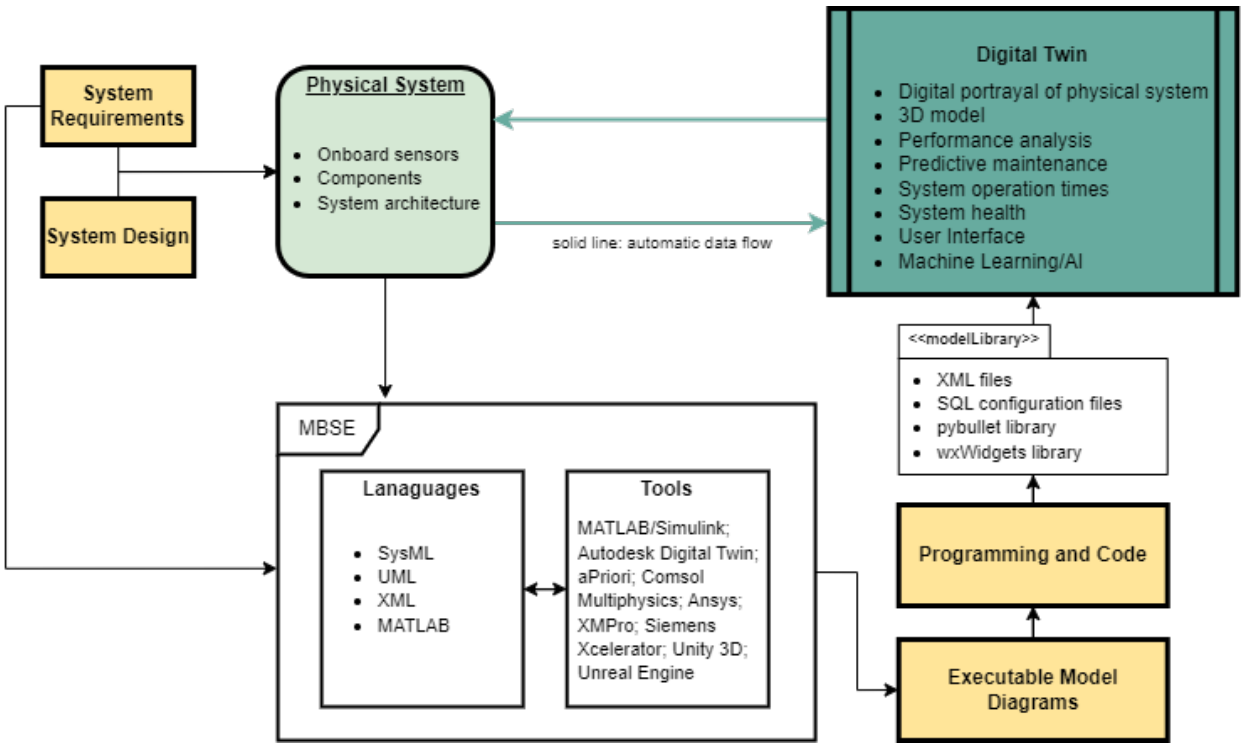

Figure 4. A Template for Developing a Digital Twin Using MBSE

MBSE-Enabled Template for Varying Virtual Models

Designing, developing, and implementing a physical system requires the definition of system requirements, which are then visualized and modeled using various MBSE modeling languages. The identification of these requirements is crucial for the development of a DT. To create complex model diagrams, modeling tools conforming to the standards of a specified modeling language can be implemented. Every change made to a feature on a diagram by a user is reflected in the specific diagram itself as well as other connected model diagrams, allowing for an organized and streamlined system design and implementation process. The most widely utilized modeling language observed in literature for developing digital twins is SysML, an extension of UML, which represents system structure, behavior, requirements, and restrictions. SysML diagrams can be categorized into nine types: block definition diagram (BDD), internal

block diagram (IBD), use case diagram, activity diagram, sequence diagram, state machine diagram, parametric diagram, package diagram, and requirements diagram. Additional information on how these SysML diagrams were observed to be used in creating digital twins and their implementation can be found in the sources listed in Table II.

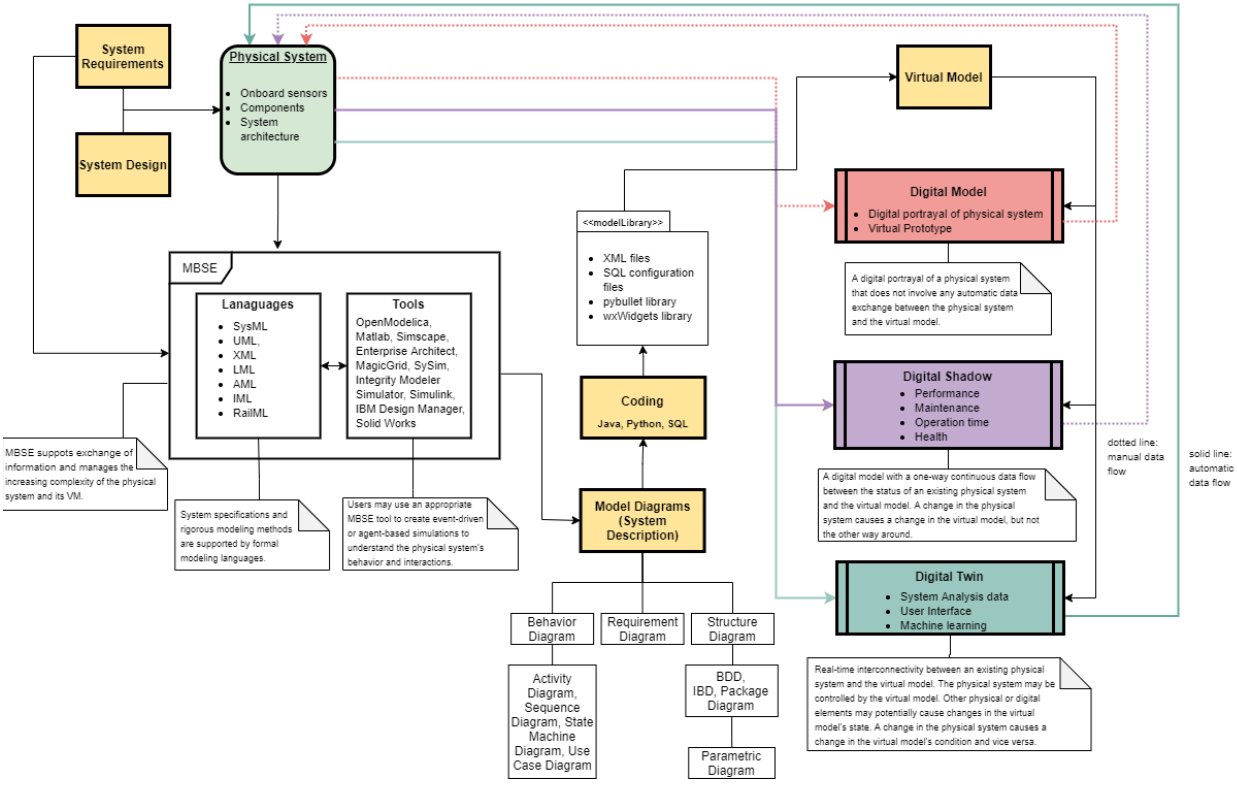

Figure 5. Developing Varying Virtual Models from a Physical System using an MBSE-Enabled Template (Lopez & Akundi, 2022).

The quantity and combination of model diagrams used to develop a virtual model type are proportional to its complexity, and the appropriate MBSE tool must be employed for optimal results. The level of MBSE integration varies across different domains, necessitating consideration of tools and data exchange components when developing the desired virtual model

type. For instance, a DM may be robust in its representation of various system components and suggest improvements, but it may not transmit data in real-time.

The defining characteristic of a digital twin (DT), regardless of the complexity of the virtual models, is its ability to transfer data in real-time to and from the physical system and virtual model, as defined by different virtual model types (Kritzinger et al., 2018). While it is possible to create digital twins without using an MBSE template, the benefits of MBSE lie in the ability to structure complicated systems and streamline the system design and implementation process. As mentioned, Table II provides an outline of the types of SysML diagrams that can be utilized to facilitate the development of virtual models, whether it be a Digital Model (DM), Digital Shadow (DS), or DT (Lopez & Akundi, 2022). MBSE tools enable the creation of virtual models with varying configurations and degrees of complexity and using SysML diagrams in combination with MBSE allows for the seamless integration of different virtual model types.

Table II. Virtual Model Types and The Corresponding SysML Model Diagram(s) Required

Digital Model	State Machine Diagram, BDD, IBD, Parametric Diagram Use Case Diagram, Activity Diagram, Requirement Diagram, and Sequence Diagram
Digital Shadow	BDD, IBD, Package Diagram, Parametric Diagram, and Requirements Diagram, Sequence Diagram, Activity diagram, Use Case Diagram, and State Machine Diagram
Digital Twin	BDD, IBD, Use Case Diagram, Activity Diagram, Sequence Diagram, State Machine Diagram, Package Diagram, and Requirements Diagram

By utilizing MBSE and SysML diagrams to develop virtual models, it is possible to enhance system performance, reduce costs, and improve the overall efficiency of the system. The ability to simulate different operating scenarios and to test various designs in a virtual environment can help to identify potential issues before they arise in the physical system, leading to improved system reliability and reduced downtime.

The subsequent step in the process involves establishing data connectivity between the physical system and the virtual model through executable program files stored in a database or model library and written in an appropriate programming language (Liu et al., 2021); (Wang et al., 2021); (Tsui et al., 2018). By reading and writing files, the system model data is processed, and the necessary information is transformed into a simulated virtual model. One way to accomplish this is by generating a SQL configuration file, retrieving database connection information from the file, and then establishing the connection when the DT starts operating. The SQL instructions are then sent to a database, and the results are stored on an MBSE modeling/simulation tool. Real-time information exchange between the physical system and virtual model is established once the query results are obtained. The type of program files created depends on the tools and system communication devices and databases utilized.

Changes in the physical system or virtual model data may be implemented manually or automatically, depending on the desired virtual model type. Figure 1 illustrates the three levels of a virtual model, which are also depicted in Figure 5. The type of virtual model generated depends on the amount and combination of model diagrams used and the method of data transfer between the physical system and virtual model. Figure 5 represents the type of data exchange through dotted or solid lines, just as in Figure 1. A colored dotted line represents manual data exchange,

where a user must manually make changes in either the physical system or virtual model, and those changes will not be reflected until the user makes the necessary adjustments manually.

On the other hand, automatic (real-time) data exchange is depicted by a colored solid line. In the case of a DS and DT, information is supplied into the virtual model type in real-time. The most significant difference between a DS and a DT is that a DT can make real-time modifications to the physical system, while a DS does not possess this capability. A more sophisticated virtual model can be created by using more advanced model diagrams and increasing the complexity of a physical system's information transmission (Lopez & Akundi, 2022).

In conclusion, to design, develop, and implement a virtual model for a physical system, a systematic approach is essential. The template specified starts with the development of system requirements before system design or employment. This approach ensures that the virtual model aligns with the needs and requirements of the physical system. System requirements can be represented using several MBSE modeling languages, with SysML identified as the most commonly used language. Once the system requirements are established, modeling tools should be used to build interconnected complex model diagrams that allow for an organized system design and implementation process. The next stage in the process is to establish data connectivity through executable program files written in a suitable programming language. These program files are stored in a database or model library and facilitate the transformation of information into a simulated virtual model once the system model data has been processed. The type of program files generated will vary depending on the tool(s) utilized, as well as the system communication devices and database(s) used. Depending on the desired virtual model type (Digital Model, Digital Shadow, or Digital Twin), information gathered from changes in the physical system or

virtual model is implemented manually or automatically. The final virtual model type is determined by the amount and combination of model diagrams utilized and the method by which data is transferred between the physical system and the virtual model. The template emphasizes the interconnectivity between different model diagrams, which allows for more coordinated and efficient system design and execution (Lopez & Akundi, 2022).

CHAPTER IV

TEMPLATE APPLICATION – A CASE STUDY ON THE APPLICATION OF MBSE FOR A UAV SURVEILLANCE SCENARIO

Scenario-Based Testing

Scenario-based testing is a powerful technique that systems engineers can use to validate the functionality of a system or product. This approach involves the creation of test scenarios that simulate different system behaviors and environments, allowing engineers to evaluate how the system performs under various circumstances (Meyer et al., 2022). Using scenario-based testing, engineers can identify potential issues and defects in the system and ensure that it meets its functional requirements. The complexity of the scenarios used in scenario-based testing will depend on the system being tested and the range of scenarios that must be evaluated. These scenarios may be simple, such as testing how the system operates under different user inputs. They may also be complex, such as evaluating the system's response to a complex set of events or environmental conditions.

The following section is divided into three distinct areas: conceptual scenario, operational scenario, and executable scenario (Figure 6). Each category has a specific purpose in characterizing the system being described (Sprockhoff et al., 2023). The operational scenario describes the system's intended use in straightforward, easy-to-understand terms. This is crucial

because it allows stakeholders who may not be familiar with technical jargon to comprehend how the system should work and its goals.

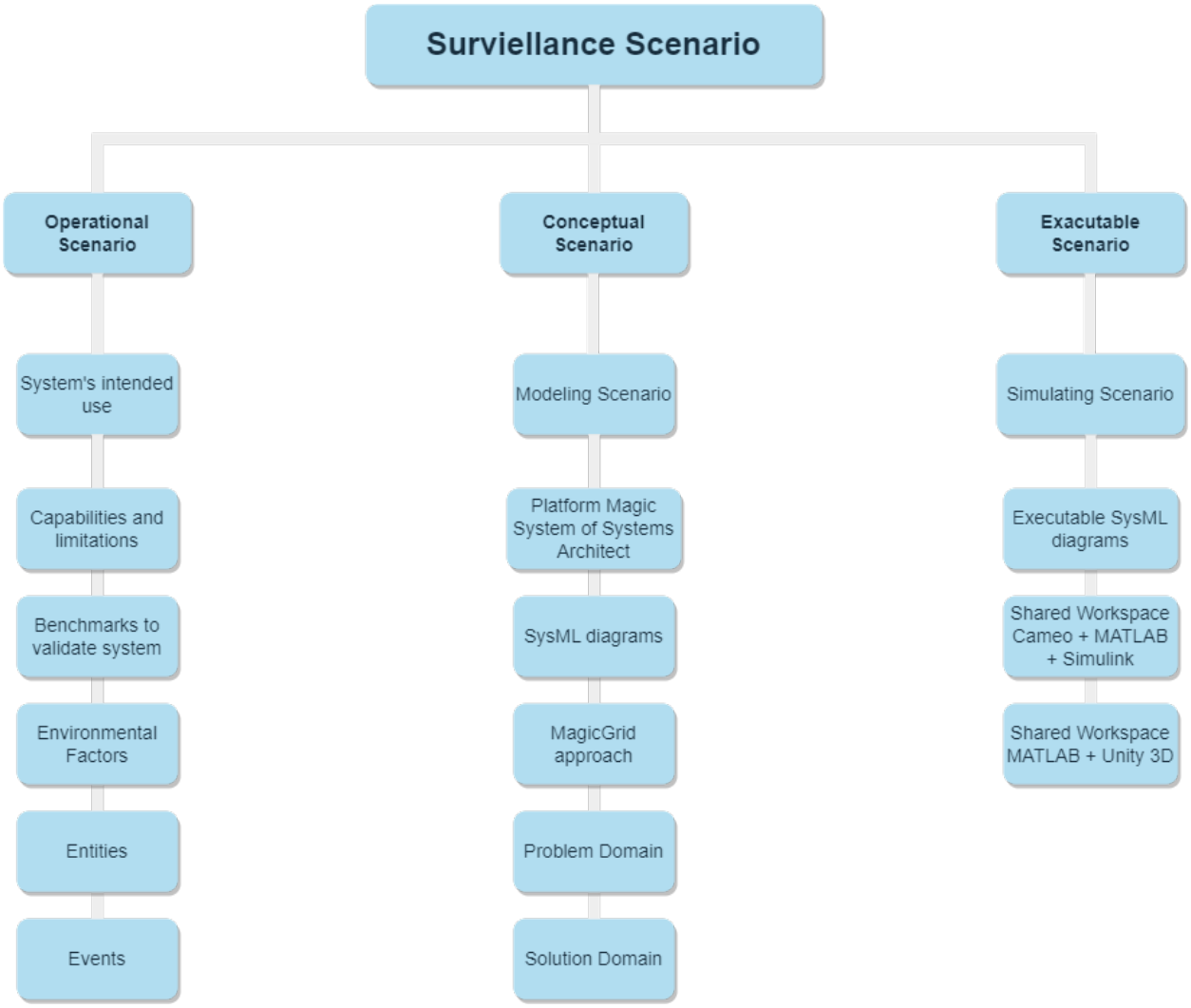

Figure 6. Breakdown of Surveillance Scenario

In contrast, the conceptual scenario formally models scenarios employing numerous aspects and their relationships. This modeling is more complicated than the operational scenario since it tries to provide a more detailed and structured knowledge of how the system will perform. This can be important for technical professionals who must comprehend the system's inner workings. Finally, the executable scenario is defined in a machine-readable format and

may be executed in a simulator. This scenario helps stakeholders observe how the system will genuinely function in practice. This tool is helpful for testing and refining the system before it is implemented. Ultimately, these three categories of scenario classification work together to provide a thorough picture of the system being described. By offering multiple viewpoints on how the system will work, stakeholders can better grasp its intended purpose and functionality.

Operational Scenario

In recent years, model-based systems engineering (MBSE) has emerged as a powerful tool for engineers to develop and validate complex systems. SysML (Systems Modeling Language) is a popular language used in MBSE, allowing engineers to model complex systems and their behaviors using a standardized notation. The Magic System of Systems Architect tool also supports MBSE by providing a collaborative platform for systems engineering teams to work together (“MagicDraw - CATIA - Dassault Systèmes®,” n.d.).

The US Army maintains a fleet of ground combat vehicles designed to undertake combat operations against opposing troops. The Congressional Budget Office has estimated the cost of such vehicles until the year 2050. The total acquisition expenditures for the Army's ground combat vehicles are estimated to average about $5 billion annually until 2050 (Congressional Budget Office. "Projected Acquisition Costs for the Army's Ground Combat Vehicles | Congressional Budget Office," n.d.). Traditionally, the Army's armored combat vehicle maintenance standards rely heavily on lengthy manual diagnostic processes (U.S. Marine Corps, 2005). Instead of using automated diagnostic paradigms, current practice only monitors if operational conditions are within the range of acceptability. There is a need for automated real-time monitoring of armored combat vehicles to evaluate ongoing vehicle health and better anticipate vehicle conditions to save both resources and lives.

Any tactical mission's objective is to defend and protect at all costs. However, maintaining and repairing tank units may be expensive and dangerous if not managed carefully and timely. To minimize servicing time and implement additional safety measures, UAV tracks and monitors a combat vehicle to detect potential changes in the tank's overall physical and structural health status and performance. The UAV will record/capture image data via an onboard camera. It will maintain a maximum altitude of two hundred meters and a minimum altitude of sixty meters from its target to maintain optimal surveillance parameters. The UAV will communicate to and from a ground control unit, as seen in Figure 7, where a flight operator can make informed decisions about the target's structural health and status from the UAV's imaging data.

The UAV will also transmit data regarding its health/battery status and performance to the operator. In the event of an abnormality in the tank's operations, the tank operator and the maintenance personnel will be notified and then equipped for unscheduled maintenance. In case of a loss of communication between the UAV and the Ground Control Unit (GCU), the UAV will continue to track and store imaging data independently, as seen in Figure 8. When a connection is lost, the GCU will alert the operator. Once the link is re-established, all stored imaging and flight data and real-time data are transmitted to the GCU. If the UAV's link is lost, it will continue to monitor and capture data from its target until the battery is down to 25% capacity and then returns to its home base.

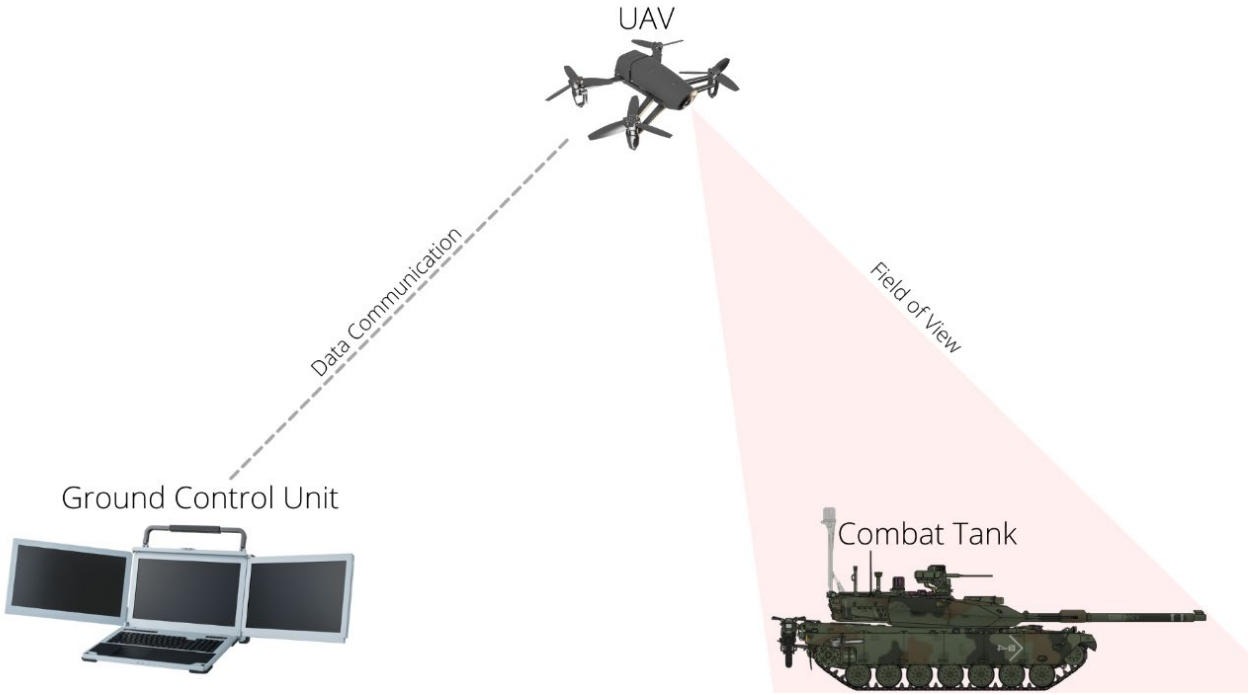

Figure 7. Surveillance Scenario

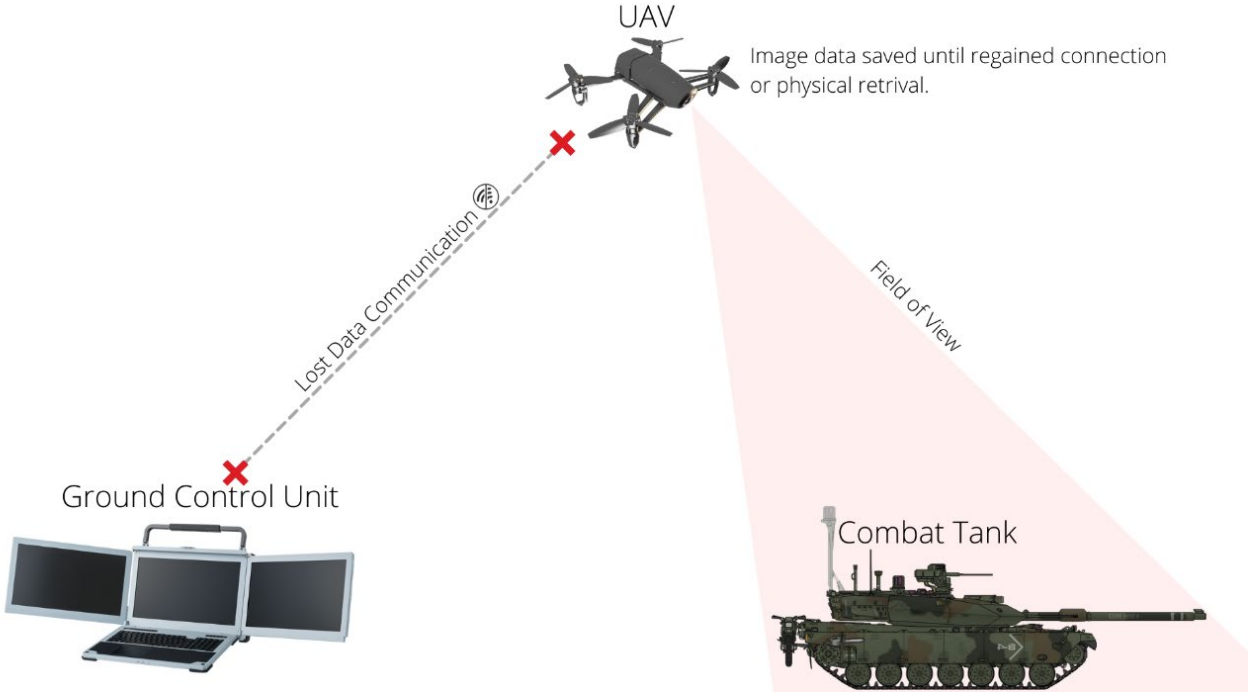

Figure 8. Lost Connection between UAV and GCU.

The modeling of the scenario will be utilized to demonstrate the feasibility of the MBSE-enabled template (Figure 5) by developing and simulating the scenario using SysML. The scenario will be modeled and simulated utilizing pre-determined optimal UAV flying parameters and weather conditions; no physical experiments were performed. It is assumed that the operator's only engagement with the surveillance systems will be assigning flight operations and analyzing incoming data.

The SysML diagrams representing the UAV surveillance scenario were created using Saulius Pavalkis' "Aircraft Radar Display SysML MagicGrid Sample with Simulation and Analysis" tutorial as a reference (Pavalkis, 2021). The tutorial provided a solid foundation and valuable insights into the use of SysML modeling techniques in the context of an aircraft radar display system. By building on this foundation, the diagrams were developed to depict the complex interactions and relationships involved in the UAV surveillance scenario.

The use of a well-established tutorial like Pavalkis' provides numerous benefits, including time-saving and increased accuracy. It allows for a more efficient development process by providing a structured approach to creating the diagrams and reducing the need for trial and error. Additionally, it ensures that the diagrams are created in accordance with established SysML standards and best practices, which helps to ensure their quality and reliability.

Furthermore, the use of a tutorial like Pavalkis' allows for the incorporation of simulation and analysis techniques in the development process. This enables the diagrams to be thoroughly tested and validated, improving their accuracy and reliability. Ultimately, the resulting diagrams provide a comprehensive representation of the UAV surveillance scenario, enabling potential stakeholders to better understand and analyze the system's behavior and functionality.

Conceptual Scenario

The UAV Surveillance Mission was divided into four categories: the problem domain, the solution domain, the UAV subsystem, and the GCU. The initial stage is to break down system information and categorize it according to what information influences each subsystem, the environment, or the mission. This is essential for simulating the transfer of imaging data, UAV health, and flight data among system elements.

Problem Domain

The UAV's health and flight data are essential system elements that need to be carefully analyzed and deconstructed into individual requirements. This allows for effective communication of information from the UAV to the GCU and human operator. To achieve this, two types of "boxes" were created, namely the Black Box and the White Box.

The Black Box is an external perspective that aims to develop a comprehensive set of requirements to prevent future revisions caused by poor specifications. It is essential for providing external insights into the system. The White Box, on the other hand, is an internal perspective that gradually identifies the system's architecture. Critical performance needs can be recorded as value attributes or flow properties in the Black Box. For instance, the system's reaction time can be described as a value property item of the Black Box that flows in or out of the system. The Black Box comprises Stakeholder requirements, Use Cases, System Context, and Performance Metrics (MoEs), as illustrated in Figure 9. On the other hand, the White Box consists of Functional Analysis, Logical architecture, and system analysis.

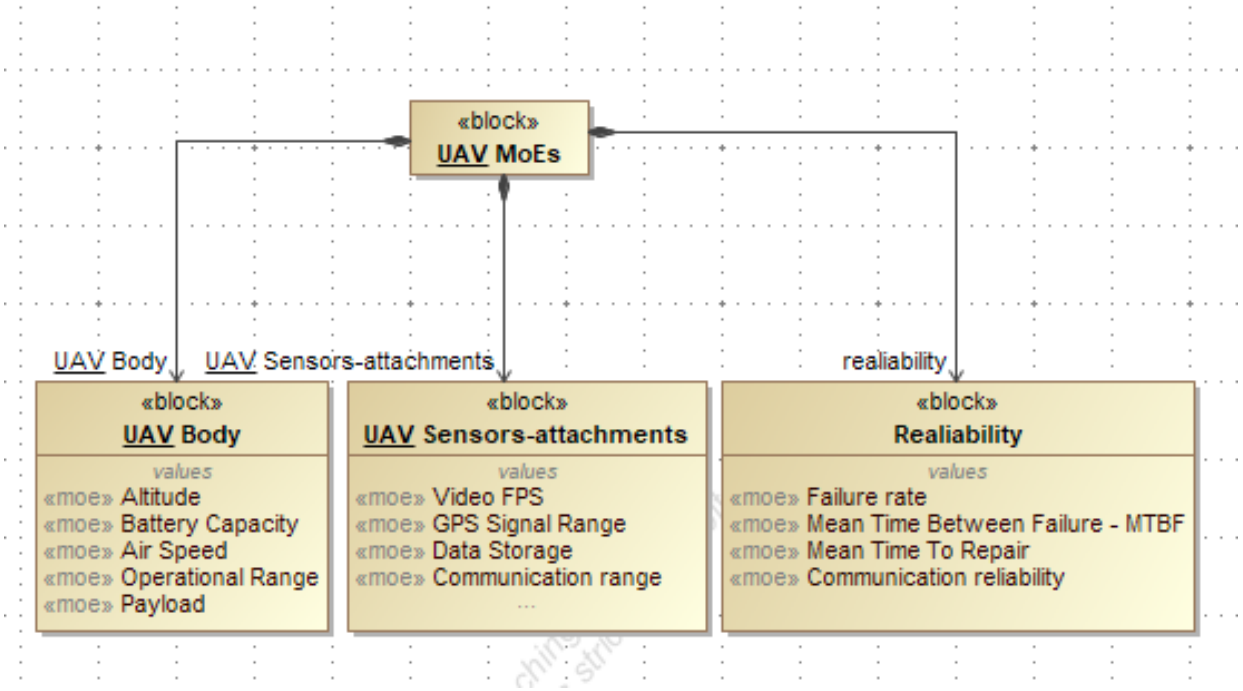

Figure 9. Functions and MoEs to Stakeholder Needs

Table III displays the requirements included in the Black Box for employing communications between the UAV and GCU. Overall, it is crucial to break down system elements into individual requirements and create both a Black Box and a White Box to achieve effective communication and avoid future revisions.

Table III. Mission Communication Requirements

1.1	Imaging data shall display in less than 1s and refresh in less than 0.5s
1.2	GCU shall support the following operation modes: pre-flight, post-flight, UAV surveillance, and warning mode.
1.3	The in-flight mode system shall display the planned trajectory of the UAV on the GCU screen.
1.4	GCU screen shall provide visual and acoustic warning in case of UAV malfunction in less than 2s

Table III, cont.

1.5	GCU screen shall provide visual and acoustic warning in case of lost connection from UAV to GCU in less than 2s
1.6	GCU screen shall provide visual and acoustic warning in case of lost connection from GCU to combat tank in less than 2s

Table III. Mission Communication Requirements Cont.

Figure 10 illustrates a comprehensive package diagram that has been employed in the development of the use case for the Ground Control Unit (GCU) operator. This diagram serves as a visual representation of the various systems and personnel involved in the operation of the GCU system. The GCU system is designed to provide critical support to the combat tank by enabling remote control of unmanned aerial vehicles (UAV) in the area of conflict. The GCU system provides a number of crucial capabilities, all of which contribute to its ability to accomplish its primary functions. This includes the ability to display all pertinent data on a screen, receive imaging data and flight data from the UAV, control the operation mode of the UAVs, and provide warnings to both the operator and the combat tank. The presentation of any significant information on a screen is critical since it gives the operator an overview of the current situation.

The GCU system can manage the flight mode of the UAVs in addition to collecting imaging and flight data from it. This gives the tank's operator the flexibility to switch between modes of operation as needed during surveillance. In addition, the operator and the combat tank may get alerts via the GCU system. This function is vital because it notifies appropriate individuals of any dangers or hazards that may develop while the UAV is in use.

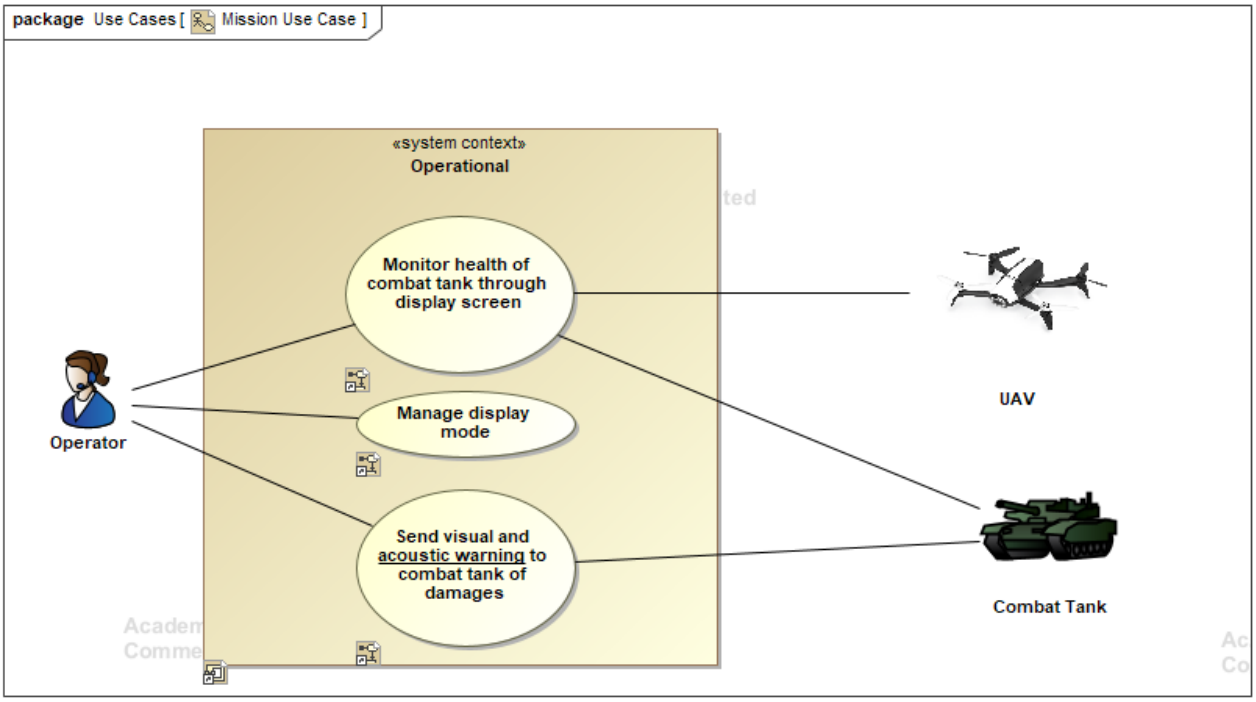

Figure 10. Operator Use Case

In order to facilitate effective communication between the subsystems, a block definition diagram (BDD) was created to define the connections between each subsystem. The BDD included ports that were referenced across multiple models, as shown in Figure 11. With these connections in place, signals such as 'location data' or 'warning' could be transmitted between subsystems (Pavalkis, 2021). To ensure that the system operated as intended, activity diagrams were constructed to represent the operator's response to information received via the GCU for each port. These diagrams, depicted in Figures 12 and 13, allowed for a clear understanding of how the operator would interact with the system in response to various inputs. The characteristics of the system's problem domain varied significantly, necessitating a comprehensive approach to problem-solving. As such, the problem domain was broken down into its various components, which were then incorporated into the solution domain (Pavalkis,

2021). This allowed for the creation of a solution that met the requirements of each individual subsystem while still functioning as a cohesive whole.

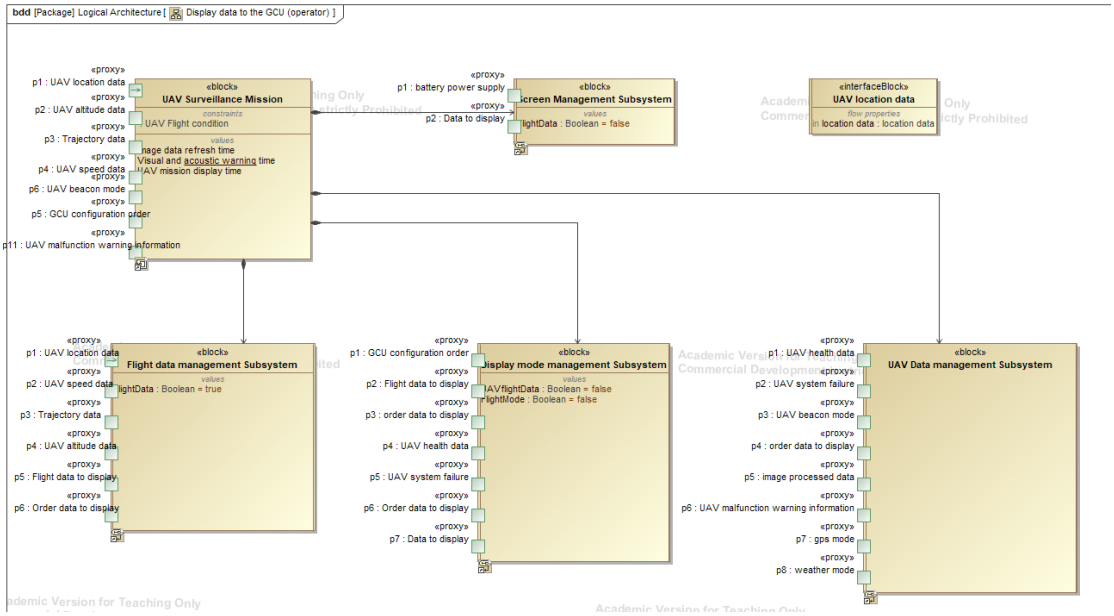

Figure 11. BDD for System Port Connection

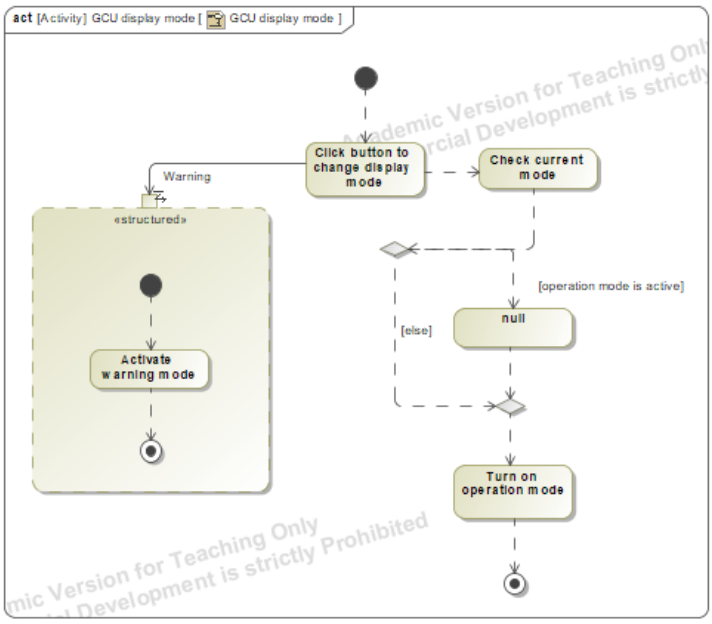

Figure 12. GCU Operation Mode Activity Diagram

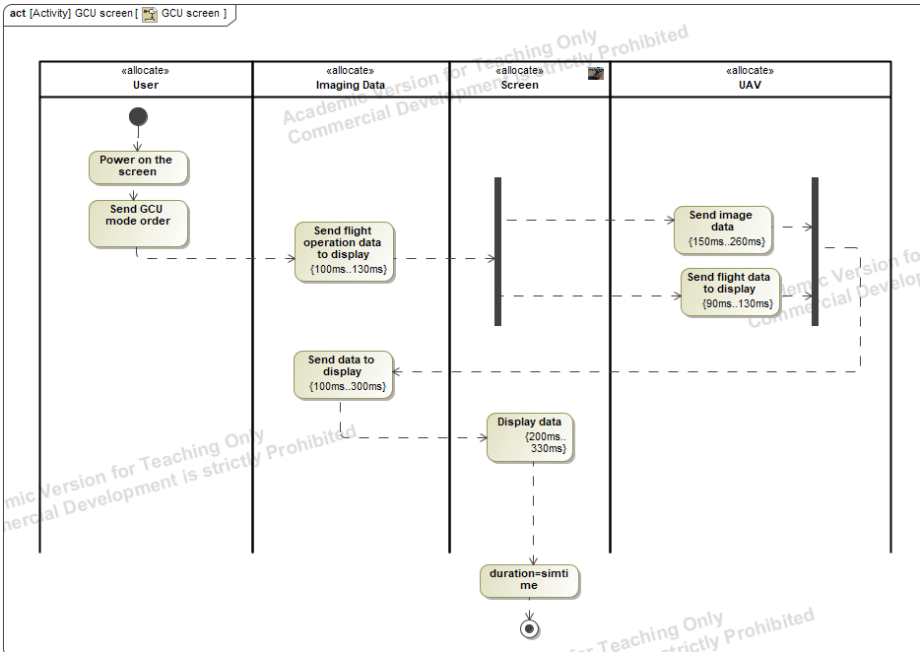

Figure 13. GCU Display Screen Activity Diagram

Solution Domain

Each activity or procedure, as well as any requirements for it, must be meticulously recorded to guarantee the system functions as planned. This documentation serves as a guide for the implementation of each subsystem, allowing for consistent and reliable operation of the overall system. The purpose of the model is to quantitatively characterize the information provided to the operator via the GCU, as depicted in Figure 14. Depending on the capabilities of the system, various amounts of data representing quantitative information, such as the duration between delivered messages or imaging data, can be transmitted.

In the subsequent phase of the project, the system's behavior is modeled within the context of the surveillance scenario. This phase employs both state and activity diagrams, allowing for greater customization and adaptability. The mission is subdivided into several states corresponding to distinct event parts, and the system states and activities are modeled to reflect

what is happening with the UAV during the operation. For example, one system state might involve ensuring that the UAV has sufficient battery life for the mission and providing a warning if it does not. For each subsystem component, an IBD was developed to characterize the relationship between the GCU, the operator, and the UAV. This diagram illustrates the transmission and reception of data between each subsystem (Pavalkis, 2021). Due to the scope of this research phase, the information was limited to the transmission and reception of imaging and flight data between subsystems. To simulate the performance of the system, data was collected on the number of milliseconds required for the UAV to transmit image data to the GCU, which the operator then uses to assess the combat tank's health. This data provides valuable insights into the system's performance and allows for the optimization of its operations.

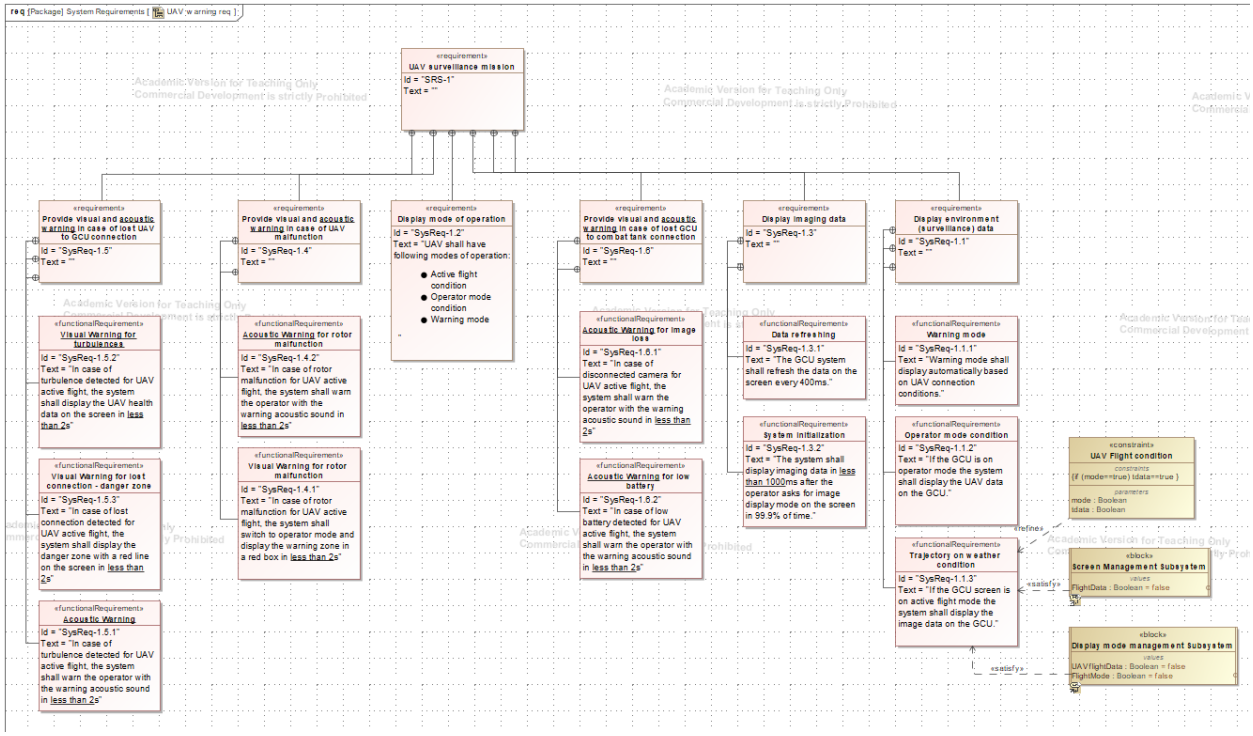

Figure 14. System Requirements

Executable Scenario

SoS-A Simulation

The scope of this research was to simulate communication between the operator, GCU, and UAV, and a duration analysis was conducted to measure the time taken (in milliseconds) for each message to be sent and displayed on the GCU, as shown in Figure 15 (Pavalkis, 2021). These results can be referenced when analyzing the previously developed GCU Display Screen Activity Diagram (Figure 13), and an example of the GCU interface and imaging data that can be sent from the UAV can be seen in Figure 16.

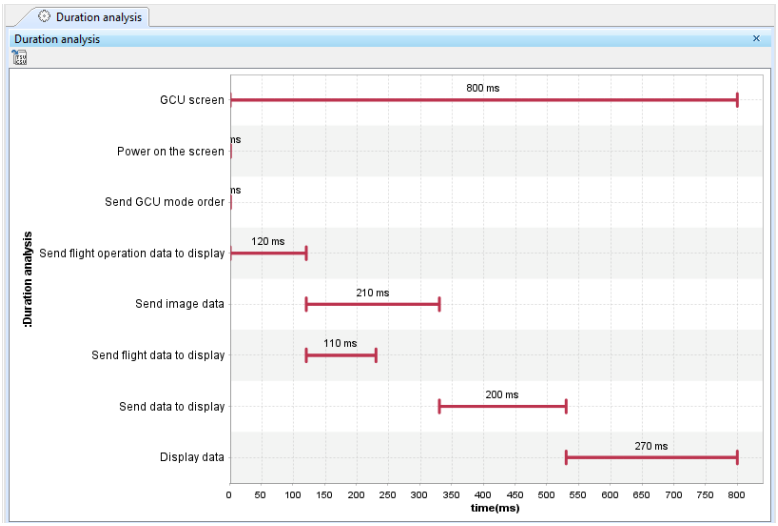

Figure 15. IBD Mission Communication Duration Times

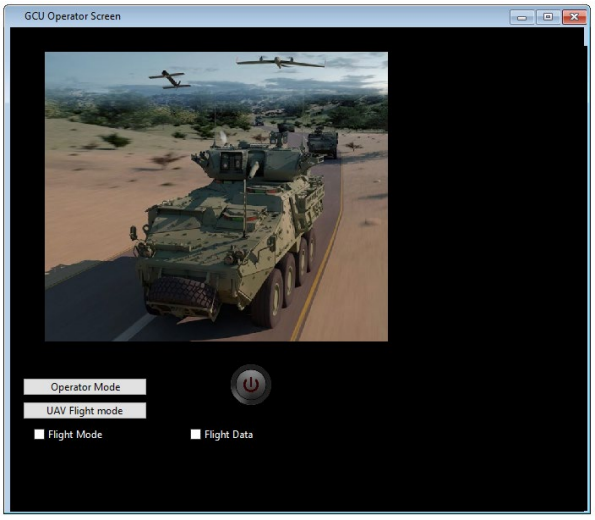

Figure 16. GCU Operator Screen with Imaging Data

Duration analysis can be a useful tool for examining several aspects of drone operations, such as flight time, battery life, and data transmission speed (Prentice, 2001). For instance, when analyzing flight time, survival analysis techniques can be used to model data on the duration of each flight, with the end of each flight serving as the "event" of interest and the duration of the flight as the "survival time". This approach can estimate the probability of a flight ending at a particular time, taking into account factors such as wind conditions, altitude, and payload weight.

Similarly, duration analysis can also be applied to examine the battery life of a drone/UAV. Data on the battery life of the drone under various operating conditions can be collected, and survival analysis techniques can be used to model the data, with the event of interest being the battery running out of charge, and the duration of battery life being the "survival time". This approach can estimate the probability of the battery running out of charge at a particular time.

Lastly, duration analysis can also be used to analyze the data transmission speed of a UAV. Data on the time it takes for the UAV to transmit distinct types of data, such as images or

sensor readings, can be collected, and survival analysis models can be used to estimate the probability of data transmission being completed at a specific time (Pavalkis, 2021). This information can be valuable in optimizing the UAV's communication system for faster and more reliable data transmission. By specifying the length of individual system behaviors in an activity diagram, duration analysis/simulation can be performed in SoS-A. Limits and ranges of time may be set, producing outcomes like maximum and random execution times (Jankevicius, 2016).

Shared Workspace

The Magic Systems of Systems Architect (SoS-A) offers a range of simulation capabilities, including four simulation engines: the Activity engine, State Machine engine, Interaction engine, and Parametric engine. For the current phase of the case study, the Parametric engine and Activity engine were utilized to model and simulate the UAV's flight sequence. Additionally, MATLAB® and Simulink® were employed to create simulations that illustrate the UAV surveillance scenario. Simulink® is a block diagram environment that supports MBSE by offering system-level design, simulation, code generation, and embedded system testing and verification. It also allows for the integration of MATLAB scripts into Simulink models (MathWorks, 2022).

Using coordinate tracking, the drone follows the combat vehicle. While traveling to a predetermined destination, the tank transmits its GPS position to the UAV. The UAV functions by maintaining a fixed distance to these coordinate positions. The operator will receive incoming flight and image data based on a time interval to verify whether the UAV is functioning correctly. For the scope of this case study, the trajectory of the combat vehicle is predetermined. Chun-Wei Kong's 6-DOF (degrees of freedom) Quadcopter Simulation and Control

MATLAB/Simulink project laid the groundwork for the simulations developed for this case study (Ahmed et al., 2022).

Using coordinate tracking, the drone tracks the movement of the combat vehicle as it travels toward a predetermined destination. The tank transmits its GPS position to the UAV, which uses this information to maintain a fixed distance from the vehicle by adjusting its own coordinates. The operator receives flight and image data at regular intervals to verify that the UAV is functioning properly.

It is worth noting that, for the purposes of this case study, the trajectory of the combat vehicle is already determined. The simulations used in this study build upon Chun-Wei Kong's 6-DOF Quadcopter Simulation and Control MATLAB/Simulink project, which provided a foundation for the development of the current simulations (Ahmed et al., 2022).

Computational Platform – MATLAB. In MATLAB scripts, flight parameters for UAV simulation were generated. SoS-A facilitates collaboration and integration between MATLAB, providing a shared workspace. While these variables can be modified in MATLAB, visualizing and defining inputs expedites model development and ensures consistency across multiple platforms and software. SoS-A recognizes expressions in MATLAB syntax, which can be modified in SoS-A and simultaneously imported into saved MATLAB files.

A block definition diagram was created to specify and visualize specific parameters in the previously established MATLAB code files. Figure 17 shows four blocks incorporated into this diagram: test, “A_SetDroneControl”, “C_XYZSignal”, and “E_animation”. The first block 'test' was constructed to verify that a shared workspace was established correctly.

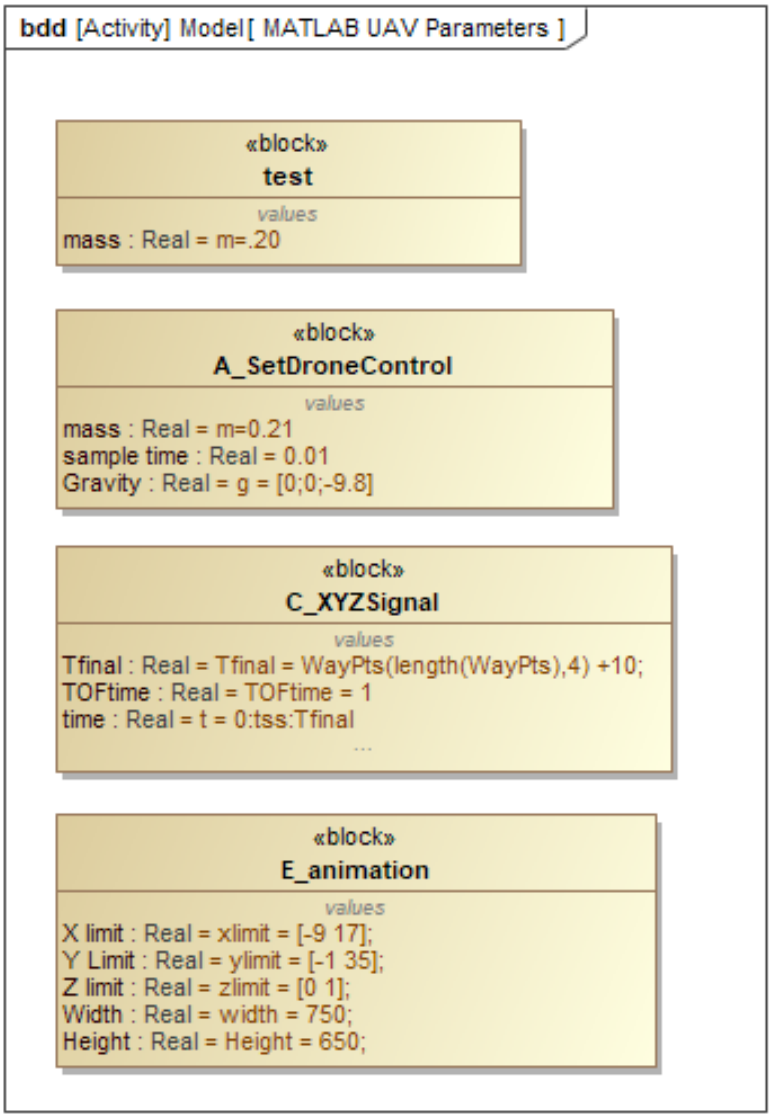

Figure 17. Block Definition Diagram, Defining MATLAB Code

As shown in Figure 18, the test block is separately chosen and simulated to verify the shared workspace. SOS-A will create a shared workspace with MATLAB after the simulation has begun, and the new mass should be represented in the corresponding file, as seen in Figure 19. This shared workspace ensures interoperability by allowing a user to make a modification on one platform and have it simultaneously updated on another. Not only does this save time, but it also ensures that all parameters, values, and inputs stay constant throughout product and system

development. Once each block has been independently simulated, resulting in updated values in the appropriate MATLAB code, an activity diagram was created to begin the required processes for executing all the MATLAB scripts to provide a simulation output.

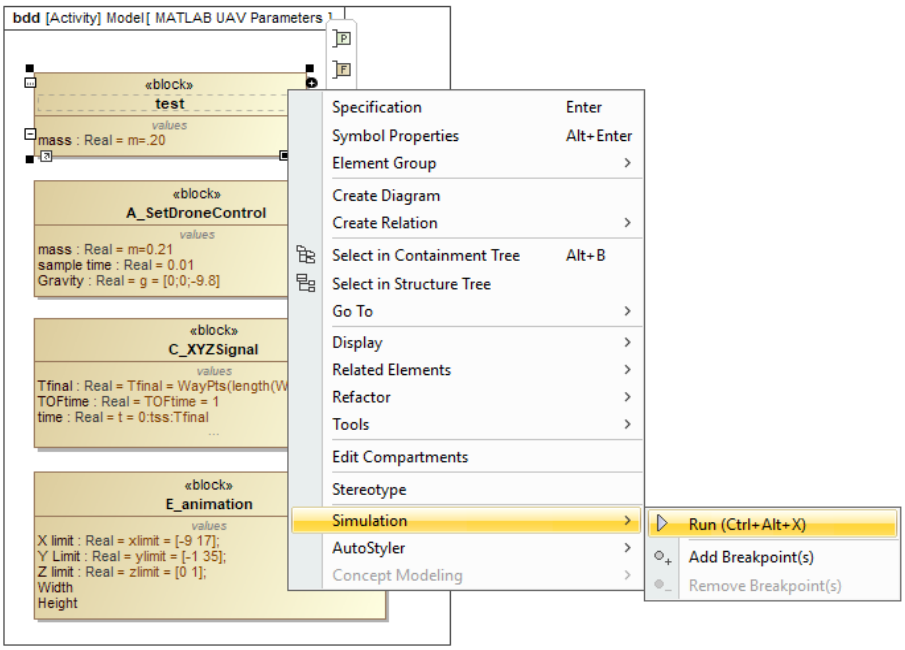

Figure 18. Simulating Test Block

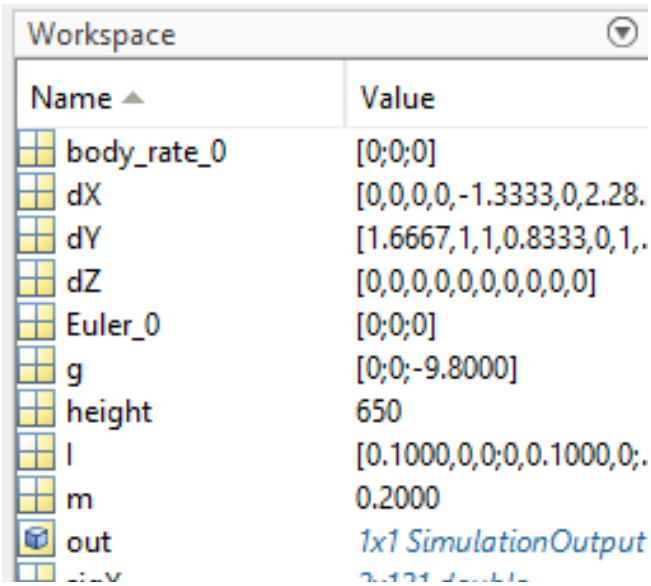

Figure 19. Updated Test Parameter

UAV 6DOF Dynamics. While SoS-A includes several simulation capabilities, the types of simulation outputs available are limited. MATLAB and Simulink provide significantly more sophisticated and dynamic simulation capabilities. When it comes to simulating the flight path of an unmanned aerial vehicle (UAV) using MATLAB and Simulink, several essential concepts exist. One of the critical concepts is using Euler angles, which refer to a set of three angles that describe the orientation of a rigid body in three-dimensional space ("6DOF (Quaternion)," n.d.). These angles are commonly used in aerospace and robotics to specify the orientation of an object or system and are denoted as roll, pitch, and yaw. Roll is the rotation around the x-axis, the pitch is the rotation around the y-axis, and the yaw is the rotation around the z-axis. It is worth noting that Euler angles can be described using different formats, such as XYZ or ZYX. The former defines the rotations in terms of successive rotations around the x, y, and z axes, while the latter defines the rotations around the z, y, and x axes ("6DOF (Quaternion)," n.d.).

The 6DOF (Euler Angles) block is used in MATLAB and Simulink to implement the Euler angle representation of six-degrees-of-freedom equations of motion (Ahmed et al., 2022). This block considers the rotation of a body-fixed coordinate frame (Xb, Yb, Zb) around a flat Earth reference frame (Xe, Ye, Ze). The block has two types of inputs, applied forces, and applied moments, both specified as a three-element vector in body-fixed axes ("6DOF (Quaternion)," n.d.). The block assumes that the applied forces act at the body's center of gravity and that the mass and inertia are constant. To use the block, you must specify several parameters, including the initial mass, body rotation rates, and Euler orientation. The primary outputs in this scenario are velocity, Euler rotation angles, and x, y, and z coordinates, although several other outputs are available.

It is important to note that the 6DOF (Euler Angles) block uses the concept of reference frames. The origin of the body-fixed coordinate frame is assumed to be the center of gravity of the body, and the body is considered rigid, eliminating the need to consider the forces acting between individual elements of mass ("6DOF (Quaternion)," n.d.). The translational motion of the body-fixed coordinate frame refers to the movement of an object in three-dimensional space, where the applied forces $\left[F_x\ F_y\ F_z\right]^T$ act within the body-fixed frame. In this scenario, the mass of the body *m* is assumed to be constant, simplifying the calculations required to determine the object's motion.

$$\overline{F_b} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m\left(\dot{\overline{V_b}} + \overline{\omega} * \overline{V_b}\right) \quad \text{("6DOF (Quaternion)," n. d.).}$$

$$\overline{V_b} = \begin{bmatrix} u_b \\ v_b \\ w_b \end{bmatrix}, \omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad \text{("6DOF (Quaternion)," n. d.).}$$

Newton's laws of motion may be used to explain an object's translational motion (Wang et al., 2016). The first law indicates that, without an external force, an object will stay at rest or in uniform motion along a straight path. The second law asserts that the rate of change of an object's momentum is proportional to the applied force, and the third rule states that every action has an equal and opposite response. In the context of the body-fixed coordinate frame, these laws can be used to describe the object's motion accurately.

To understand the relationship between the body-fixed angular velocity vector, $[p\ q\ r]^T$, and the rate of change of the Euler angles, $\left[\dot{\phi}\ \dot{\theta}\ \dot{\psi}\right]^T$, it is important to understand the concept of Euler rates and the body-fixed coordinate frame. Again, the Euler angles are a set of three angles that describe the orientation of a rigid body in three-dimensional space. These angles represent

rotations around three orthogonal axes. The body-fixed angular velocity vector, $[p\ q\ r]^T$, describes the angular velocity of the body about these three axes.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

(Wang et al., 2016).

Resolving the Euler rates into the body-fixed coordinate frame is a necessity for analyzing the connection between these two variables. This entails translating the Euler rates, which are measured relative to an Earth-centered, Earth-fixed reference frame, into the coordinate frame of the body. After resolving the Euler rates into the body-fixed coordinate frame, the relationship between the body-fixed angular velocity vector and the rate of change of the Euler angles can be determined. This relationship can be expressed mathematically using the equations of motion for a rigid body, which describe how the angular velocity of a body changes in response to external forces and moments (“6DOF (Quaternion),” n. d.); (Wang et al., 2016).

Instead of attempting to build a simulation output for the flight path of the UAV in SoS-A, an activity diagram was created to develop a shared workspace with MATLAB and Simulink. Using Cameo Simulation Toolkit, it is possible to call MATLAB/Simulink functions directly from Magic Systems of System Architect. MATLAB is one of the supported evaluation tools. When invoking MATLAB functions, UML/SysML model parameters can be input and run MATLAB/Simulink models returning results to the SysML models. After the proper parameters have been imported using the previously described block definition diagram, the necessary MATLAB and Simulink files can be loaded via the activity diagram to provide a simulation output

of the UAV flight route. Figure 20 depicts the five files that must be executed/loaded to complete the simulation.

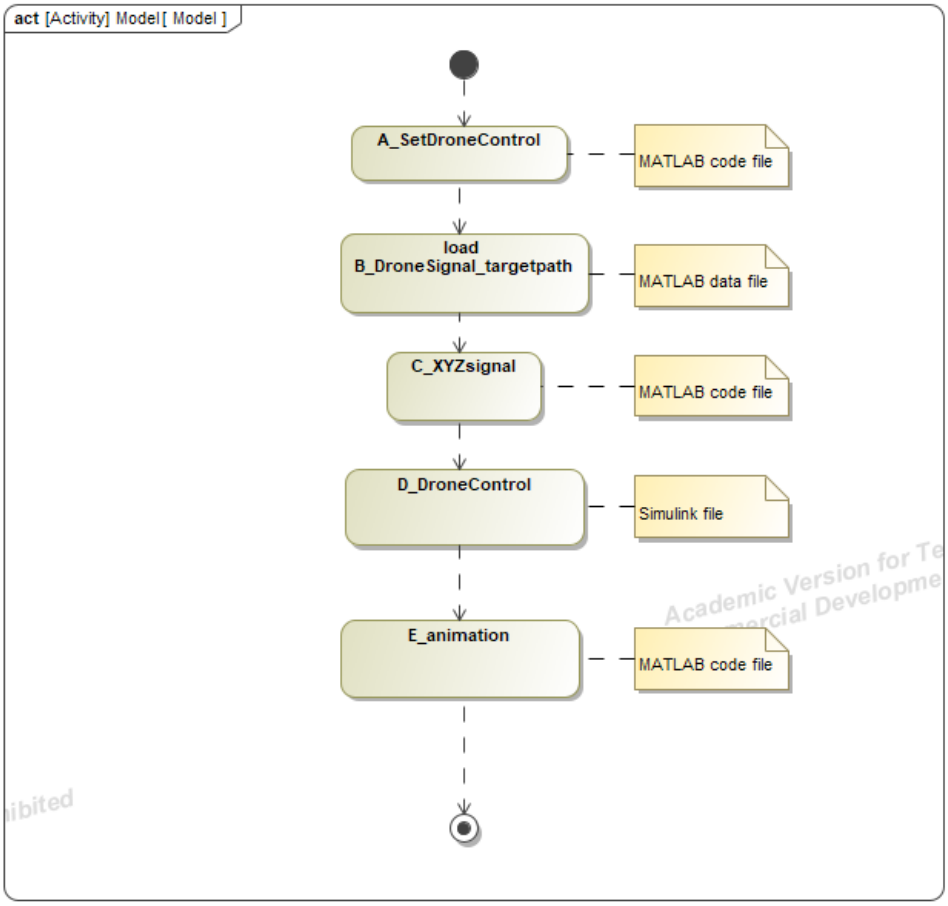

Figure 20. MATLAB-Simulink Simulation Activity Diagram

'A_SetDroneControl' initializes the UAV's parameters. The 'B_DroneSignal' target path determines the coordinates the UAV must travel to follow the target by determining the X, Y, and Z points and T (time) required to reach each set of coordinates using a matrix (Ahmed et al., 2022). The 'C_XYZsignal' MATLAB script is then executed to calculate the UAV's velocity and Euler's angles at each set of points specified. The results of these computations are then input into Simulink File 'D_DroneControl', and output values are subsequently sent into 'E_animation'. The activity diagram will execute these files chronologically and depict the current step (Figure

21). Figure 22 depicts the simulation's successful conclusion after the stages have been concluded.

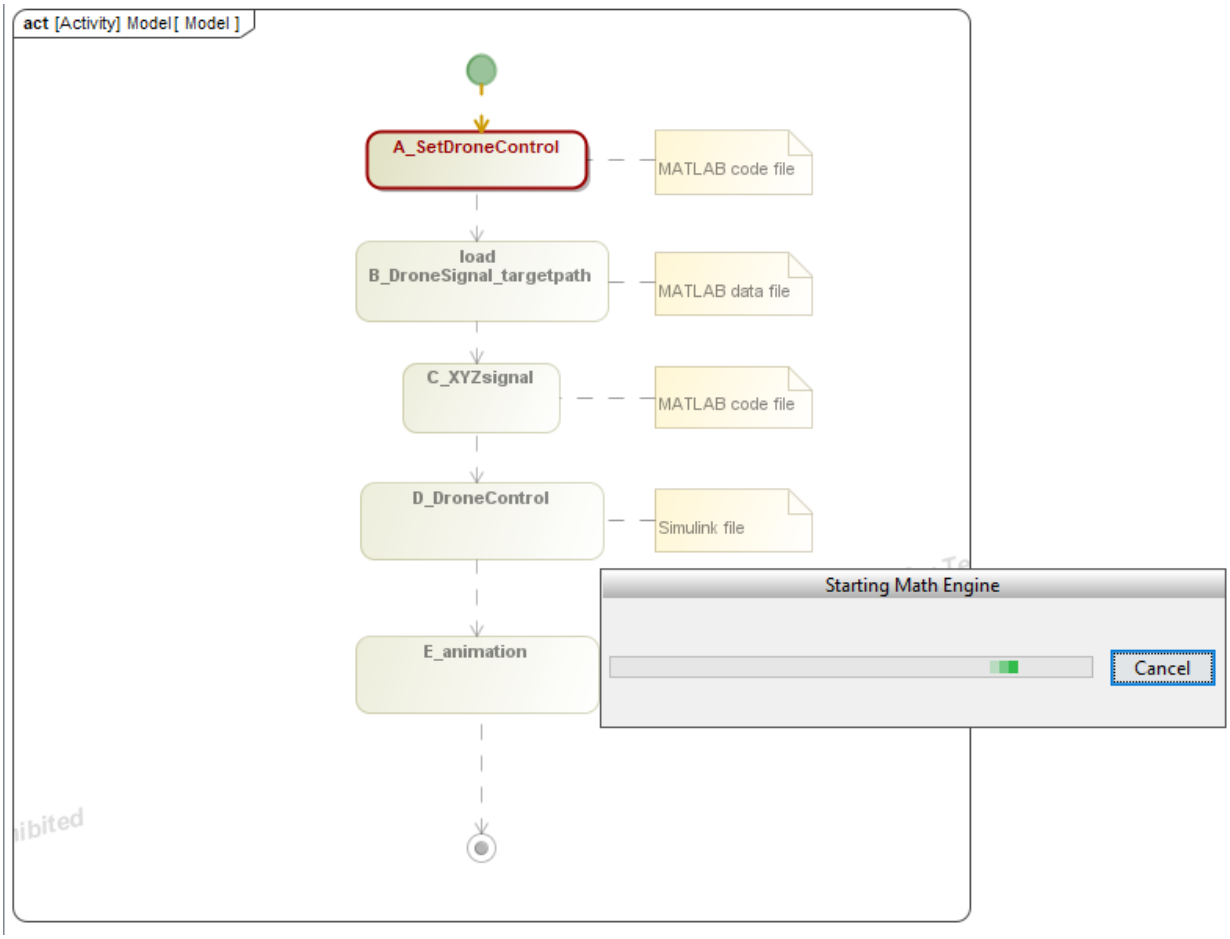

Figure 21. Simulating Activity Diagram and Establishing Shared Workspace

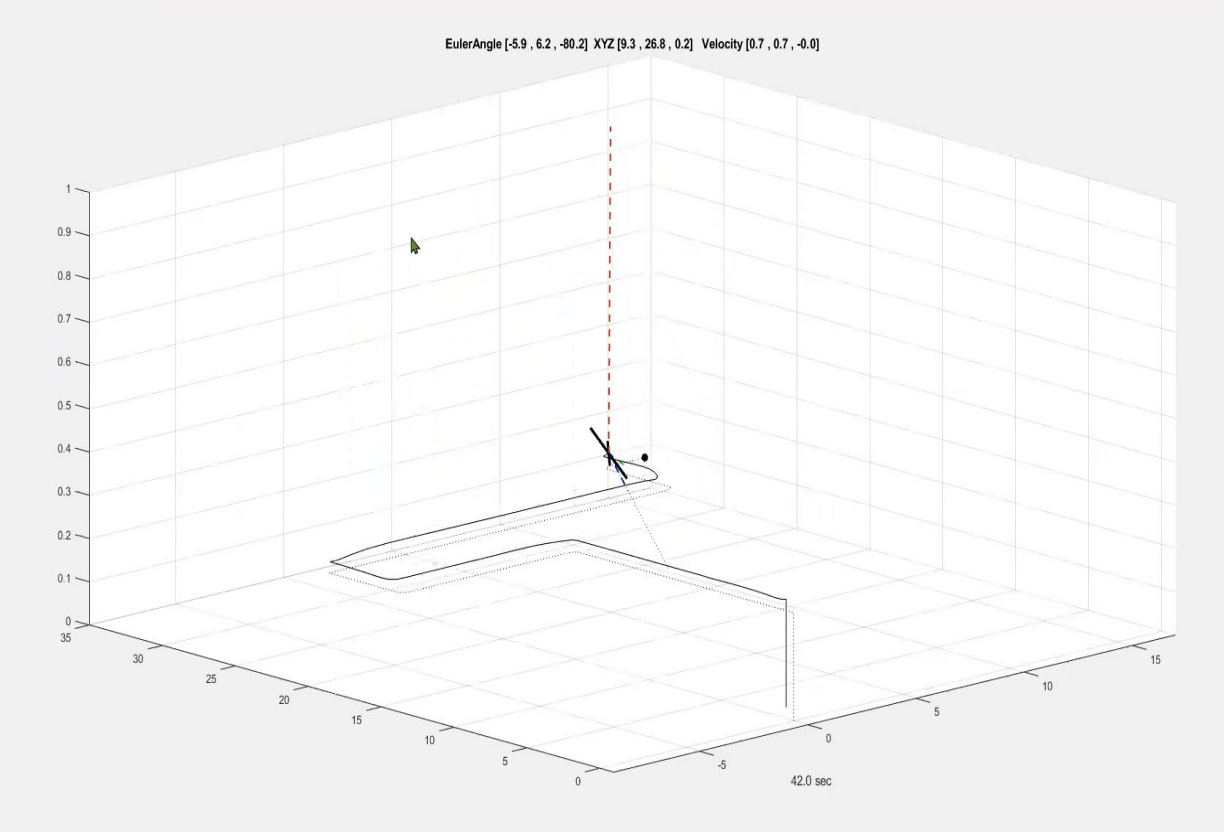

Figure 22. UAV Flight Path Simulation

SoS-A, MATLAB, and Simulink's unified workspace enhance simulation capabilities. Due to the number of files and input parameters required for the simulation of the UAV flight route, errors can occur. SoS-A guarantees that the required input values are visualized and maintained in MATLAB and Simulink if user or system requirements change. It, moreover, makes it simple for other users to duplicate the actions required to achieve the simulation outcome. A user with no prior MATLAB knowledge may simply execute the corresponding files using the created shared workspace. SoS-A indicates to users and stakeholders that the architecture satisfies system and stakeholder requirements. The models and shared workspace that SoS-A develops allow for the representation of a system that assures user comprehension.

MBSE aims to provide a framework for models that makes them comprehensible and manageable throughout the development of complex systems.

Visualization. While both SoS-A and MATLAB offer several simulation features, there are limits regarding visualization. Using a game engine is intended to result in more dynamic simulations. Using algorithms and scripting languages, a gaming engine such as Unity 3D may significantly improve simulations. Unity 3D, or Unity for short, is a popular game engine that enables 2D and 3D visual effects. In Unity, it is possible to recreate any scenario or system; nonetheless, the notion of a cross-platform shared workspace is essential to this research. Given Unity's numerous APIs, C# script is primarily employed for this project's scope. Following the integration of SoS-A and MATLAB, the next objective is integrating Unity into this shared workspace.

Getting Started with Unity. Unity offers varied options for individuals, teams, and businesses; a free student plan was used for this project. The Unity Hub and Unity Editor were installed on a Windows 10 system. Microsoft Visual Studio 2022 was used to develop C# scripts and manage repositories and packages. Due to Unity's built-in physics engine, the same simulation parameters used in MATLAB will also be used in this environment. A real-time link between MATLAB and Unity must be validated before beginning work in Unity. To feed data from MATLAB into Unity, MATLAB was set up as a client, and Unity was set up as a server. MATLAB's *tcpclient* command generates a TCP/IP client that synchronizes to a server connected with the requested host IP address and port (“Object Creation Properties,” n.d.). In order to use a port, a number between 1 and 65535 must be used. The corresponding inputs set both the Address and Port properties. Figure 23 depicts the C++ code written to establish MATLAB as the client.

```
clc
clear all
tcpipClient = tcpclient("127.0.0.1",55001,"Timeout",20,"ConnectTimeout",30)
fopen(tcpipClient);
a='connection successful!';
fwrite(tcpipClient,a);
fclose(tcpipClient);
```

Figure 23: C++ Code for Client Set Up

"tcpclient(address, port, Name, Value)" establishes a connection using name-value pair arguments. *The address* is the IP address, the *port* is the remote host port, *Timeout* is the allowed time to conclude processes, and *ConnectionTimeout* is the allowed time to connect to the remote host.

Like tcpclient, TcpListener is a class that needs a local IP address and port number to admit incoming connection requests ("Object Creation Properties," n.d.). The same IP address and port number from Figure 23 are used in Figure 24 to establish a connection between MATLAB and Unity. Methods Pending() checks for the presence of any outstanding connection requests where AcceptTcpClient() responds to said requests. C# script must then be written to establish Unity as a server, as seen in Figure 24. The C# file must be dragged and dropped onto the Main Camera in the Unity editor. The message "Unity is listening" is seen in the Unity console and awaits the message "connection successful!". Once this message was seen in the console (Figure 25), the real-time connection between the two platforms was confirmed. However, to visualize changes, the script in MATLAB had to be executed multiple times. A User Datagram Protocol (UDP) was then utilized and imported into the Simulink file used in Figure 26. A UDP is a network interface that allows programs to communicate across the internet with minimal delay and interruption ("Basic UDP Communication - MATLAB & Simulink

Example," n.d.). UDP speeds up data transfers by allowing data to be sent before the receiving side makes an agreement.

```
public class readSocket : MonoBehaviour
{
    TcpListener listener;
    String msg;
    void Start()
    {
        listener = new TcpListener(IPAddress.Parse("127.0.0.1"), 55001);
        listener.Start();
        print("Unity is listening");
    }

    void Update()
    {
        if (!listener.Pending())
        {
        }
        else
        {
            print("socket comes");
            TcpClient client = listener.AcceptTcpClient();
            NetworkStream ns = client.GetStream();
            StreamReader reader = new StreamReader(ns);
            msg = reader.ReadToEnd();
            print(msg);
        }
    }
}
```

Figure 24: C# Script For Establishing Unity as A Server

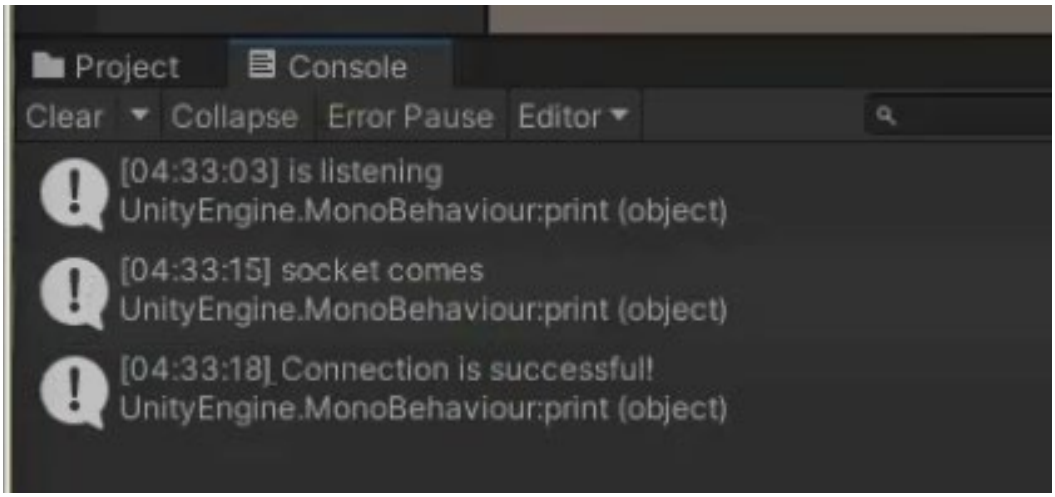

Figure 25: Successful Connection Between MATLAB and Unity

The set of coordinates specified in 'B_DroneSignal_targetpath.mat' is the input signal supplied to the UDP Send block (Figure 26) via a Simulink File (Figure 27). The 'Remote IP address' is used by the UDP Send and Receive blocks to facilitate communication between the two platforms.

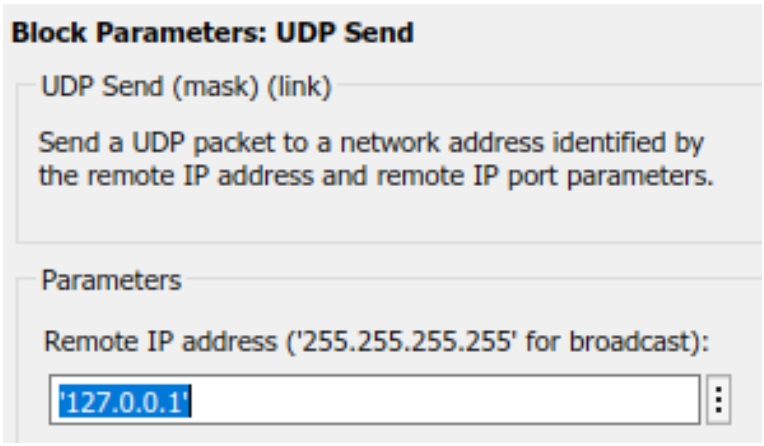

Figure 26: UPD Send Block

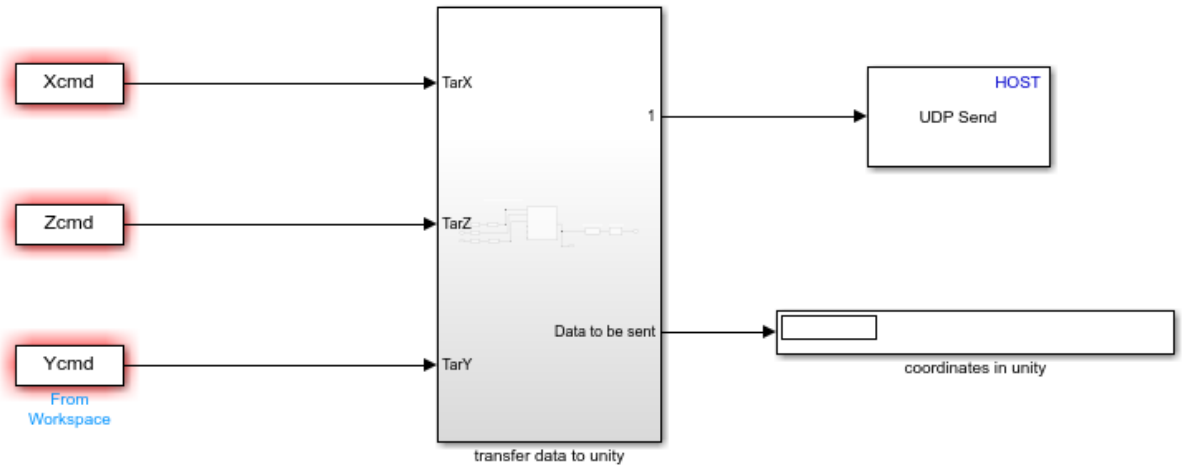

Figure 27: Simulink Model for MATLAB-Unity Shared Workspace

The three data inputs seen in Figure 27 must first be loaded in the MATLAB workspace before the Simulink file can successfully run simultaneously with Unity. Now that files have successfully been configured in MATLAB, in order to get the UPD block to communicate with Unity, a C# script must be generated and attached to the combat tank (target). Using *transformPosition* (Figure 28), the target will move to each of the coordinates specified in 'B_DroneSignal_targetpath.mat'.

```
void Update()
{
    transform.position = new Vector3(transformPosition[0], transformPosition[1], transformPosition[2]);
    transform.rotation = Quaternion.Euler(transformRotation[0], transformRotation[1], transformRotation[2]);
}
```

Figure 28: C# Script for UAV Target Positioning.

The Unity engine employs the left-handed cartesian coordinate system, wherein clockwise rotation around the axis of rotation is positive. Within Unity, there are two basic coordinate systems: local and global. Local coordinates represent a game object's location relative to another, while global coordinates represent a game object's position inside the overall

space of the Unity project. Now that the MSOS target parameters and coordinates have been linked with MATLAB and Unity, a more comprehensive simulation may be developed.

Unity Capabilities. The Unity Asset Store offers diverse free and purchasable 2D and 3D models, templates, and tools to speed and aid game development ("The Best Assets for Game Making," n.d.). A free demo pack created by Jonah Hessel, including low-poly PBR ready WW2-era tanks, was utilized for this scenario (Hessel, n.d.). More specifically, the SOMUA_S35 model tank was used (Figure 29). This game object is the inheritor of the coordinates previously defined. Any object could easily be replaced with SOMUA_S35 and still follow the movement and positioning defined in MATLAB. However, this model tank was utilized for the scope of this scenario.

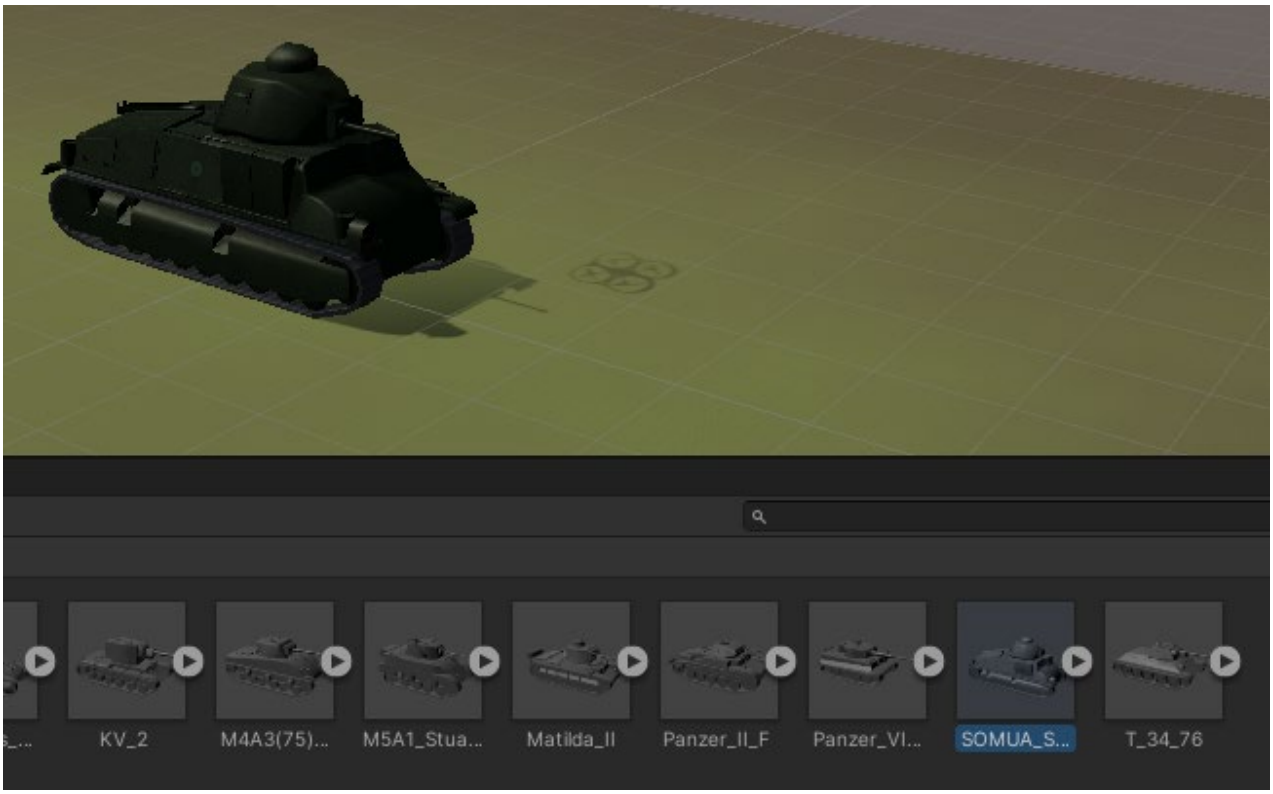

Figure 29: SOMUA_S35 Model Tank in Unity World

The UAV for this scenario is fashioned via a modification of a free drone controller demo created by Mario Haberle, accessible from the Unity Asset Store ("FPV Drone Controller: Physics," n.d.). Due to the requirement for perfect weather conditions for UAV operation, all other game elements, such as terrain and environment, are cosmetic. The UAV's positioning and movement depend on the tank's positioning and movement. The main objective of the UAV is to maintain a safe distance from the tank as it moves so that it may collect imaging data (Figure 30). A camera is included to visualize said imaging data.

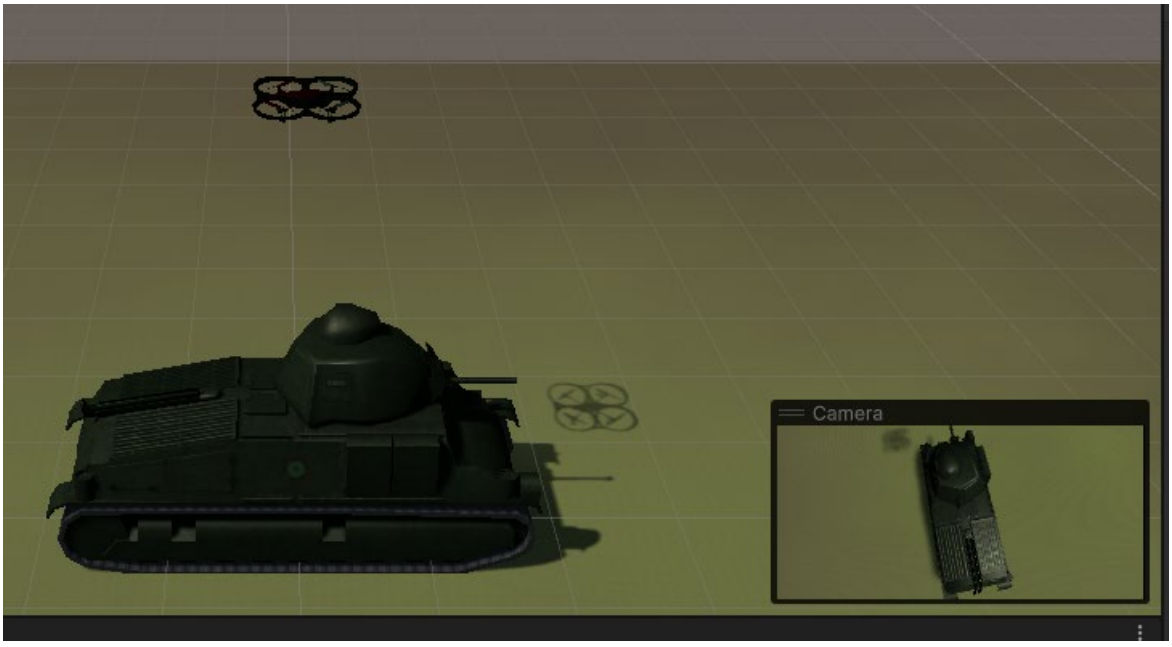

Figure 30: UAV Target Scenario

Another C# script is required for the UAV to autonomously follow the tank while keeping a safe distance (Figure 31). Using *transform.position* and *relativePosition,* the UAV camera will move relative to the parent game object (combat tank). Similarly, the UAV body will move relative to the game parent object using *transform.position* and the specified distance

and speed parameters (Figure 32). In addition to moving toward the target, the UAV's body will rotate, so the target is always inside the camera's field of view (FOV).

```
public class cameraFollow : MonoBehaviour
{
    public float smoothness;
    public Transform target;
    private Vector3 initalOffset;
    private Vector3 cameraPosition;

    public enum RelativePosition { InitalPosition, Position1, Position2 }
    public RelativePosition relativePosition;
    public Vector3 position1;
    public Vector3 position2;

    void Start()
    {
        relativePosition = RelativePosition.InitalPosition;
        initalOffset = transform.position - target.position;
    }

    void FixedUpdate()
    {
        cameraPosition = target.position + CameraOffset(relativePosition);
        transform.position = Vector3.Lerp(transform.position, cameraPosition, smoothness * Time.fixedDeltaTime);
        transform.LookAt(target);
    }
```

Figure 31: C# Script for UAV Camera

```
ic class droneFollow : MonoBehaviour

public float MinDistance = 15;
public float MaxDistance = 45;
public float Speed = 10;
public Transform target;


void Update()
{
    if (Vector3.Distance(transform.position, target.position) >= MinDistance)
    {
        Vector3 follow = target.position;
        follow.y = this.transform.position.y;

        this.transform.position = Vector3.MoveTowards(this.transform.position, follow, Speed * Time.fixedDeltaT
    }
}
```

Figure 32: C# Script for UAV Body

The camera settings can also be easily changed within Unity. Local coordinates determine the position, rotation, and scale of the camera. The FOV axis is set to vertical, and the

FOV ranges from 1e-05 to 179. The type of camera sensor ranges from 8mm to 70mm, as seen in Figure 33.

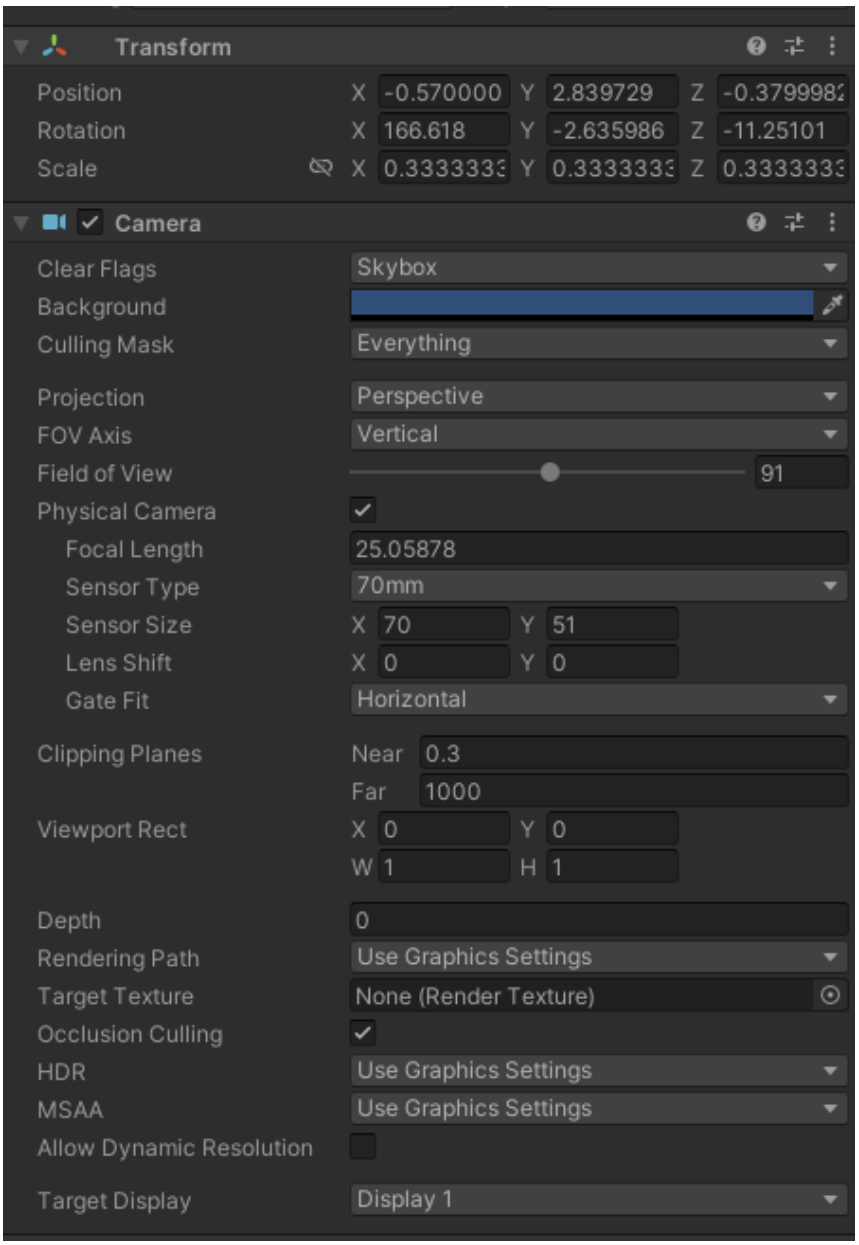

Figure 33: UAV Camera Settings

Unity contains numerous simulation capabilities. Due to the limited experience with Unity and the limited scope of this research, Unity was used for visualization purposes. Using Unity, a more dynamic model, or even a true digital twin, can be created.

CHAPTER V

DISCUSSION

Limitations and Gaps in Related Work

Data Sources

The objective of this research was to explore a hypothetical scenario involving an unmanned aerial vehicle (UAV) conducting surveillance on a combat battle tank, specifically for a military operation. Given the lack of access to actual physical UAV flight data, the data inserted into the SysML diagrams were approximations. Even with access to actual physical data, the time and resources required to develop virtual models would have been significantly greater. Data sources coming from a physical system must first be recorded and verified. This data must then be consistently communicated to the developed/developing virtual model. There is a need for standardized methods that expedite, streamline, and integrate the MBSE methods and tools previously mentioned. However, the purpose of this study is to provide an example of procedures, tools, and modeling languages that can be applied to model, simulate, and analyze a true physical system after its deployment.

Executable Models

Jasper Sprockhoff et al., implement an MBSE approach to develop an AI-based system (Sprockhoff et al., 2023). Using SysML diagrams, a threat localization system for aircraft object

detection is modeled. They model how their AI-based system should detect potential encounters and notify the level of danger. While they are able to effectively model their system and create and executable implementation outside of SysML diagrams, there is a disconnect from their simulation and modeling platform.

Similarly, authors Bajaj et al., determine the gaps in *current state-of-the-art tools for design and simulation of complex systems* (Bajaj et al., 2011). They detail a disconnect between the different lifecycle phases of a system. Gap 1 pertains to the absence of model-based continuity in system design and simulation activities that extend from the initial design stages to subsequent design stages (Bajaj et al., 2011). Gap 2 pertains to the discrepancies that arise between design and analysis/simulation models during various design stages (Bajaj et al., 2011). For instance, this gap can appear between conceptual system design models and mathematically based analysis models in the initial stages of design.

Chapter III presents a template for digital model development that is enabled by MBSE, with the objective of resolving the issues mentioned above. Since the publication by Bajaj et al., there has been a noteworthy expansion in the availability of MBSE collaborative tools. The application of the Model-Based Systems Engineering (MBSE) enabled template facilitated the establishment of a collaborative workspace among three distinct platforms for modeling, simulating, and visualization. Additionally, it addressed the research of Jasper Sprockhoff et al.'s absence of executable SysML diagrams. In addition to being used to import UAV variables like speed and timing, SysML activity diagrams were generated and executed to initiate the simulations carried out in MATLAB. The simulations conducted in MATLAB were visualized more effectively by integrating Unity 3D into the shared workspace of SoS-A and MATLAB.

Virtual Model Types

Each platform utilized in this work can be directly traced to the templates shown in Figures 2, 3, and 4. SoS-A was used to create a digital model of the UAV surveillance scenario. SoS-A data was then imported into MATLAB/Simulink in real time to create a digital shadow. With input from MATLAB and Simulink, a digital twin can then be created in Unity 3D using real-time data connection. While the work shown here cannot be considered a digital twin in the strict sense due to the lack of a physical system, it does demonstrate the streamlined capabilities required to construct a digital twin from a physical one.

Challenges and Lessons Learned

Due to the multidisciplinary nature of MBSE, an authentic digital twin cannot be created by a single individual. The created templates demonstrate the processes required for generating digital models which can potentially be used as a digital twin's research testbed. Different types of software knowledge are required by both system designers and end users in the development of virtual models. When a person is able to focus their efforts on a single platform for the modeling, simulation, or visualization of a system, that system's dependability may improve. The developed templates are meant to be used by a team of system designers and stakeholders. MBSE enables this traceability between not only tools but different system users.

Integration of the three distinct applications required extensive troubleshooting. SoS-A allows for the use of four distinct simulation engines: the activity engine, the state machine engine, the interaction engine, and the parametric engine. Due to an absence of access to real-world data, both the activity engine and parametric engine were utilized in SoS-A simulations. The parametric engine allowed for the simulation of blocks including UAV variables like mass

and gravity constant, with the data subsequently being sent on to MATLAB. The activity engine was also used to connect and execute MATLAB files that were developed outside of SoS-A. Despite SoS-A's compatibility with MATLAB script syntax generation, it was decided that testing script would be more beneficial in a separate environment.

Unity integration with MATLAB required setting up a client-server connection between the two programs. When Unity was first set up as the server and MATLAB as the client, the waypoints would not transfer across properly. To make the integration work, MATLAB had to be set up as a client, while Unity had to be set up as a server. The sample time was another factor that affected how the MATLAB simulation was displayed. Adjustments had to be made to lengthen the time it took the UAV to reach each coordinate point so that it could clearly be depicted in the Unity environment. There was some temporal discrepancy between the two simulations, but the combined tank and UAV Unity simulation appropriately maintained the coordinates set up in MATLAB.

CHAPTER VI

CONCLUSION

This research investigates the viability of utilizing Model-Based Systems Engineering (MBSE) alongside SysML to design, model, and simulate a surveillance system to track, record, and communicate information on an armored combat vehicle's state of health and performance through an unmanned aerial aircraft (UAV). In order to build SysML-compliant models and produce state machine diagrams and activity diagrams, the Magic System of Systems Architect (SoS-A) platform was used (Pavalkis, 2021). This research evaluates the efficacy of the processes from the developed conceptual template facilitated by MBSE for creating virtual models (Lopez & Akundi, 2021). SysML was used to represent the UAV surveillance scenario, with simulation performed in MATLAB and Simulink. The scenario was broken down into four distinct classes. The results of the research demonstrate that MBSE-based modeling aids in system visualization, organization, assessment, and validation.

Furthermore, the Magic Systems of Systems Architect is a simulation-supporting tool. SoS-A has four different simulation engines: Activity, State Machine, Interaction, and Parametric. In this work, the Parametric and Activity engines were used to simulate the flight path of an unmanned aerial vehicle (UAV). Also employed to demonstrate the scenarios were MATLAB and Simulink, with Simulink functioning as a block diagram environment utilizing Model-Based Systems Engineering (MBSE). MATLAB scripts have been generated

(Ahmed et al., 2022) to simulate the UAV flight parameters, and in SoS-A, activity and block diagrams were utilized to integrate MATLAB and provide a collaborative workspace. The shared workspace improves simulation capabilities and guarantees that input values are retained if user or system requirements are modified. This shared workspace offers an approach for models that makes them understandable and controllable throughout the development of complicated systems on multiple platforms.

Future Work

This work involved employing Unity as a virtual environment for the unmanned aerial vehicle scenario. As mentioned, the Unity Hub and Unity Editor were installed on a Windows computer, while Microsoft Visual Studio 2022 was utilized for C# script creation. A real-time connection was created between MATLAB and Unity using the tcpclient and TcpListener commands. The coordinates from a MATLAB file were inserted into a UDP Send block so that the two platforms could communicate. Then, a C# script was written to manage the UAV and target's movement and placement. The Unity Asset Store was employed for game features like tanks and UAVs, and a C# script was created enabling the UAV to follow the tank safely. As a result of its varied APIs and packages, Unity offers unmatched simulation capabilities. More game objects can be incorporated to enhance the complexity of this scenario. Introducing UAV object detection and avoiding inclement weather circumstances may highlight Unity's physics engine better. In this scenario, object animations were kept minimal. The next steps include animating individual components on the UAV and combat vehicle, like rotors and tracks. Developing an interactable graphical user interface (GUI) to emulate the ground control station is a new objective. SoS-A, MATLAB, Simulink, and eventually Unity would all benefit from importing data from a real-world physical system. Each new element enhances the simulation's

dynamic in Unity, bringing the digital representation one step closer to becoming a true digital twin.

REFERENCES

Ahmed, S., Qiu, B., Kong, C. W., Xin, H., Ahmad, F., & Lin, J. (2022). A Data-Driven Dynamic Obstacle Avoidance Method for Liquid-Carrying Plant Protection UAVs. *Agronomy*, *12*(4), 873.

Akundi, A., & Lopez, V. (2021). A review on application of model based systems engineering to manufacturing and production engineering systems. *Procedia Computer Science*, *185*, 101-108.

Anyanhun, A. I., & Edmonson, W. W. (2018, April). An MBSE conceptual design phase model for inter-satellite communication. In *2018 Annual IEEE International Systems Conference* (*SysCon*) (pp. 1-8). IEEE.

Bachelor, G., Brusa, E., Ferretto, D., & Mitschke, A. (2019). Model-based design of complex aeronautical systems through digital twin and thread concepts. *IEEE Systems Journal*, *14*(2), 1568-1579.

Bajaj, M., Zwemer, D., Peak, R., Phung, A., Scott, A. G., & Wilson, M. (2011, March). Slim: collaborative model-based systems engineering workspace for next-generation complex systems. In *2011 Aerospace Conference* (pp. 1-15). IEEE.

Bretz, L., Tschirner, C., & Dumitrescu, R. (2016, October). A concept for managing information in early stages of product engineering by integrating MBSE and workflow management systems. In *2016 IEEE International Symposium on Systems Engineering (ISSE)* (pp. 1-8). IEEE.

Carnegie Mellon University. (n.d.). *Architecture analysis and Design Language (Aadl)*. Architecture Analysis and Design Language (AADL). Retrieved April 3, 2023, from https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=191439%2C191439

Congressional Budget Office. (2021, April). *Ground combat vehicles - congressional budget office*. Projected Acquisition Costs for the Army's Ground Combat Vehicles | Congressional Budget Office. Retrieved April 9, 2022, from https://www.cbo.gov/system/files/2021-03/57085-ground-combat-vehicles.pdf

Delbrügger, T., & Rossmann, J. (2019). Representing adaptation options in experimentable digital twins of production systems. *International Journal of Computer Integrated Manufacturing*, *32*(4-5), 352-365.

Delligatti, L. (2013). *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley.

Eclipse. (n.d.). *Features and benefits*. Capella MBSE Tool - Features. Retrieved April 6, 2023, from https://www.eclipse.org/capella/features.html

Eclipse. (n.d.). *Let yourself be guided with Arcadia*. Capella MBSE Tool - Arcadia. Retrieved April 6, 2023, from https://www.eclipse.org/capella/arcadia.html#:~:text=Arcadia%20is%20a%20model%2Dbased,all%20the%20Thales%20business%20domains.

Feiler, P. H., Gluch, D. P., & Hudak, J. J. (2006). *The architecture analysis & design language (AADL): An introduction*. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.

Friedenthal, S., Griego, R., & Sampson, M. (2007, June). INCOSE model based systems engineering (MBSE) initiative. In *INCOSE 2007 symposium* (Vol. 11). sn.

Glatt, M., Sinnwell, C., Yi, L., Donohoe, S., Ravani, B., & Aurich, J. C. (2021). Modeling and implementation of a digital twin of material flows based on physics simulation. *Journal of Manufacturing Systems*, *58*, 231-245.

Grieves, M., & Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. *Transdisciplinary perspectives on complex systems: New findings and approaches*, 85-113.

Haberle, M. (n.d.). *FPV Drone Controller: Physics*. Unity Asset Store. Retrieved November 10, 2022, from https://assetstore.unity.com/packages/tools/physics/fpv-drone-controller-118390

Hause, M. (2019, July). The Digital Twin Throughout the SE Lifecycle. In *INCOSE International Symposium* (Vol. 29, No. 1, pp. 203-217).

Henderson, K., & Salado, A. (2021). Value and benefits of model-based systems engineering (MBSE): Evidence from the literature. *Systems Engineering*, *24*(1), 51-66.

Hessel, J. (n.d.). *Making 3D models for the Unity Asset Store*. JonahH. Retrieved November 10, 2022, from https://unity.everthessel.nl/.

IBM. (n.d.). *Engineering systems design rhapsody - overview*. IBM Engineering Systems Design Rhapsody. Retrieved April 1, 2023, from https://www.ibm.com/products/systems-design-rhapsody

Jankevicius, N. (2016, November 22). Webinar – Time and Duration Analysis. [PowerPoint slides]. MBSE Product Manager, No Magic, Inc. https://blog.nomagic.com/timing-duration-analysis/

Kritzinger, W., Karner, M., Traar, G., Henjes, J., & Sihn, W. (2018). Digital Twin in manufacturing: A categorical literature review and classification. *Ifac-PapersOnline*, *51*(11), 1016-1022.

Lee, B. K. E. (2021). *Enhancing Mission Engineering Route Selection Through Digital Twin Decision Support* (Doctoral dissertation, Monterey, CA; Naval Postgraduate School).

Lemazurier, L., Chapurlat, V., & Grossetête, A. (2017). An MBSE approach to pass from requirements to functional architecture. *IFAC-PapersOnLine*, *50*(1), 7260-7265.

Li, L., Soskin, N. L., Jbara, A., Karpel, M., & Dori, D. (2019). Model-based systems engineering for aircraft design with dynamic landing constraints using object-process methodology. *IEEE Access*, *7*, 61494-61511.

Liu, J., Liu, J., Zhuang, C., Liu, Z., & Miao, T. (2021). Construction method of shop-floor digital twin based on MBSE. *Journal of Manufacturing Systems*, *60*, 93-118.

Liu, S., Bao, J., Lu, Y., Li, J., Lu, S., & Sun, X. (2021). Digital twin modeling method based on biomimicry for machining aerospace components. *Journal of manufacturing systems*, *58*, 180-195.

Lopez, V., & Akundi, A. (2022, April). A conceptual model-based systems engineering (mbse) approach to develop digital twins. In *2022 ieee international systems conference (syscon)* (pp. 1-5). IEEE.

Madni, A. M., & Purohit, S. (2021, October). Augmenting MBSE with Digital Twin Technology: Implementation, Analysis, Preliminary Results, and Findings. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 2340-2346). IEEE.

Madni, A. M., Madni, C. C., & Lucero, S. D. (2019). Leveraging digital twin technology in model-based systems engineering. *Systems*, *7*(1), 7.

MagicDraw - CATIA - Dassault Systèmes®. "MagicDraw - CATIA - Dassault Systèmes®." www.3ds.com. Accessed May 13, 2022. https://www.3ds.com/productsservices/catia/products/no-magic/magicdraw/.

MathWorks. (n.d.). *6DOF (Quaternion)*. MathWorks. Retrieved March 10, 2023, from https://www.mathworks.com/help/aeroblks/6dofeulerangles.html.

MathWorks. (n.d.). *Basic UDP Communication*. Basic UDP Communication - MATLAB & Simulink Example. Retrieved April 1, 2023, from https://www.mathworks.com/help/instrument/basic-udp-communication.html

MathWorks. (n.d.). *Object Creation Properties.* MathWorks. Retrieved March 10, 2023, from https://www.mathworks.com/help/matlab/ref/tcpclient.html.

MathWorks. (n.d.). Simulink - simulation and model-based design. Simulation and Model-Based Design - MATLAB &. Retrieved September 14, 2022, from https://www.mathworks.com/products/simulink.html

Meyer, M. A., Silberg, S., Granrath, C., Kugler, C., Wachtmeister, L., Rumpe, B., ... & Andert, J. (2022, June). Scenario-and Model-Based Systems Engineering Procedure for the SOTIF-Compliant Design of Automated Driving Functions. In *2022 IEEE Intelligent Vehicles Symposium (IV)* (pp. 1599-1604). IEEE.

No Magic, Inc. (n.d.). *Magic Systems of Systems Architect Documentation*. No Magic Product Documentation. Retrieved April 1, 2023, from https://docs.nomagic.com/display/MSOSA2022xR1/Magic+Systems+of+Systems+Architect+Documentation

Object Management Group. (n.d.). *ABOUT THE OMG SYSTEM MODELING LANGUAGE SPECIFICATION VERSION 1.7 BETA*. About the OMG System Modeling Language Specification version 1.7 beta. Retrieved April 2, 2023, from https://www.omg.org/spec/SysML/

Object Management Group. (n.d.). *What is UML*. What is UML | Unified Modeling Language. Retrieved April 2, 2023, from https://www.uml.org/what-is-uml.htm

Pang, T. Y., Pelaez Restrepo, J. D., Cheng, C. T., Yasin, A., Lim, H., & Miletic, M. (2021). Developing a digital twin and digital thread framework for an 'Industry 4.0'Shipyard. *Applied Sciences*, *11*(3), 1097.

Pavalkis, S. (2021). *Aircraft Radar Display SysML MagicGrid Sample with Simulation and Analysis*. *YouTube*. YouTube. Retrieved April 2, 2022, from https://www.youtube.com/watch?v=JtWZQM-yamk&t=559s.

Phanden, R. K., Sharma, P., & Dubey, A. (2021). A review on simulation in digital twin for aerospace, manufacturing and robotics. *Materials today: proceedings*, *38*, 174-178.

Pirnstill, C. (2022). Human Digital Twin and Modeling Guidebook.

Prentice, R. L., & Kalbfleisch, J. D. (2001). Survival Analysis: Overview.

Sage, A. P., & Rouse, W. B. (2014). *Handbook of systems engineering and management*. John Wiley & Sons.

Sakairi, T., Palachi, E., Cohen, C., Hatsutori, Y., Shimizu, J., & Miyashita, H. (2013). Model based control system design using SysML, Simulink, and computer algebra system. *Journal of Control Science and Engineering*, *2013*, 9-9.

Schluse, M., Priggemeyer, M., Atorf, L., & Rossmann, J. (2018). Experimentable digital twins—Streamlining simulation-based systems engineering for industry 4.0. *IEEE Transactions on industrial informatics*, *14*(4), 1722-1731.

Sheard, S. A., & Mostashari, A. (2009). Principles of complex systems for systems engineering. *Systems Engineering*, *12*(4), 295-311.

Sillitto, H., Martin, J., McKinney, D., Griego, R., Dori, D., Krob, D., ... & Jackson, S. (2019, September). Systems engineering and system definitions. In *INCOSE*.

Sprockhoff, J., Lukic, B., Janson, V., Ahlbrecht, A., Durak, U., Gupta, S., & Krueger, T. (2023). Model-Based Systems Engineering for AI-Based Systems. In *AIAA SCITECH 2023 Forum* (p. 2587).

Staskal, O., Simac, J., Swayne, L., & Rozier, K. Y. (2022, June). Translating SysML Activity Diagrams for nuXmv Verification of an Autonomous Pancreas. In *2022 IEEE 46th*

Annual Computers, Software, and Applications Conference (COMPSAC) (pp. 1637-1642). IEEE.

Tschirner, C., Dumitrescu, R., Bansmann, M., & Gausemeier, J. (2015, April). Tailoring Model-Based Systems Engineering concepts for industrial application. In *2015 Annual IEEE Systems Conference (SysCon) Proceedings* (pp. 69-76). IEEE.

Tsui, R., Davis, D., & Sahlin, J. (2018, July). Digital Engineering Models of Complex Systems using Model-Based Systems Engineering (MBSE) from Enterprise Architecture (EA) to Systems of Systems (SoS) Architectures & Systems Development Life Cycle (SDLC). In *INCOSE International Symposium* (Vol. 28, No. 1, pp. 760-776).

U.S. Marine Corps. (2018, April 4). *Marine Corps Tank Employment*. Retrieved from https://www.marines.mil/portals/1/Publications/MCTP%203-10B.pdf?ver=2019-03-21-140858-470

Unity. (n.d.). *The best assets for game making*. Unity Asset Store. Retrieved November 10, 2022, from https://assetstore.unity.com/.

Wang, H., & Ma, D. (2016, July). Aircraft 6-DOF Modular Modeling Based on MATLAB Simulink. In *2nd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2017)* (pp. 1002-1005). Atlantis Press.

Wang, Y., Steinbach, T., Klein, J., & Anderl, R. (2021). Integration of model based system engineering into the digital twin concept. *Procedia CIRP*, *100*, 19-24.

APPENDIX

APPENDIX

TERMS AND DEFINITIONS

Systems Engineering (SE) - Systems engineering considers every step of a system's life cycle, from design and development through retirement, to solve problems. Systems engineering is a transdisciplinary and integrative approach that enables the successful realization, use, and retirement of engineered systems through the application of systems principles (Sillitto et al., 2019).

System Of Systems (SoS) - The term "System of Systems" (SoS) refers to a system that integrates several smaller systems into one larger system in order to perform a specific operation.

Model-Based Systems Engineering (MBSE) - Model-based systems engineering (MBSE) is the systematic use of models from the early stages of a project's conceptual design all the way through its development and validation at the end of its life cycle, as defined by the 2007 INCOSE Model-Based Systems Engineering Initiative (Friedenthal et al., 2007).

Unified Modeling Language (UML) - The Unified Modeling Language (UML) is a set of diagrams that have been standardized to aid in the specification, visualization, construction, and documentation of software system artifacts (“What is UML,” n.d.).

Systems Modeling Language (SysML) - Systems Modeling Language, or SysML for short, is a graphical modeling language for creating, analyzing, and defining complex systems. Using the notations and diagrams provided by SysML, system architectures may be represented in a more thorough and structured fashion. SysML has nine distinct types of diagrams (Delligatti, 2013).

Architecture Analysis and Design Language (AADL) - Modeling language AADL (Architecture Analysis and Design Language) was developed specifically to describe the structure of real-time and embedded systems. It is used to formally analyze the attributes of the system, such as timing and resource use, by representing the structure and behavior of the system at various levels of abstraction ("Architecture analysis and Design Language (Aadl)," n.d.).

Digital Twins (DT) - Virtual copies, or "digital twins," are created by simulating a physical system in a computer. DTs use information gathered from sensors, simulations, and other sources to provide engineers a real-time look into the system's performance, allowing for continuous monitoring and optimization. (Grieves, 2017).

Digital Model (DM) - A DM is a digital depiction of a physical system that does not utilize any computerized data exchange between the physical system and the virtual model (Kritzinger et al., 2018).

Digital Shadow (DS) - A DS is a digital depiction of an integrated one-way data flow between the state of an existing physical system and the state of a virtual model (Kritzinger et al., 2018).

Magic System of Systems Architect (SoS-A) - Magic System of Systems Architect (SoS-A) is a No Magic, Inc. developed software application used for designing and analyzing complex systems of systems ("Magic Systems of Systems Architect Documentation," n.d.).

MATLAB/Simulink - MATLAB is a programming environment for engineers and scientists to evaluate and build systems and products. Simulink, a MATLAB add-on, allows interactive, graphical modeling, simulation, and analysis of dynamic systems. MATLAB and Simulink let you simulate your system using textual and graphical programming (MathWorks, 2022).

Unity – Unity 3D, or Unity, is a robust, cross-platform 3D engine with an intuitive development environment.

UAV - An unmanned aerial vehicle (UAV), sometimes known as a drone, is an aircraft that has no human pilot, crew, or passengers on board.

GCU - The central command and control base for remotely piloted aircraft is the ground control unit (GCU).

MoEs - MoEs is an abbreviation for Measures of Effectiveness. MoEs evaluate tactical objectives but not strategic or operational ones.

BIOGRAPHICAL SKETCH

The author, Viviana Guadalupe Lopez, was born and raised in the Rio Grande Valley. Their permanent mailing address is 605 North O Street, Harlingen, Texas, 78550. However, the best way to initiate contact is via email (vivianlopez92@live.com). In the spring of 2021, the author graduated from The University of Texas Rio Grande Valley with a bachelor's degree in Engineering Technology. They also worked as an undergraduate research assistant during this time. The University of Texas Rio Grande Valley's College of Engineering and Computer Science awarded her a prestigious honor, the Presidential Research Fellowship. This award was effective from Fall 2021 to Spring 2023 and allowed her to continue her studies toward a master’s degree in Manufacturing Engineering (MSE) which was conferred in May of 2023.