Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

3-2022

Geometric Generation Through the Merging of SYsML, and Engineering Sketch Pad

Alexander J. Miesle

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Systems Engineering Commons

Recommended Citation

Miesle, Alexander J., "Geometric Generation Through the Merging of SYsML, and Engineering Sketch Pad" (2022). *Theses and Dissertations*. 5415.
https://scholar.afit.edu/etd/5415

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact AFIT.ENWL.Repository@us.af.mil.

GEOMETRIC GENERATION THROUGH MERGING OF SYSML AND ENGINEERING SKETCH PAD

THESIS

Alexander J. Miesle, Captain, USAF

AFIT-ENV-MS-22-M-237

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENV-MS-22-M-237

AUTOMATED GEOMETRIC GENERATION THROUGH SYSML AND ENGINEERING SKETCH PAD

THESIS

Presented to the Faculty

Department of Systems Engineering and Management

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Systems Engineering

Alexander J. Miesle, BS

Captain, USAF

March 2022

DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENV-MS-22-M-237

AUTOMATED GEOMETRIC GENERATION THROUGH SYSML AND ENGINEERING SKETCH PAD

Alexander J. Miesle, BS

Captain, USAF

Committee Membership:

Dr. Thomas C. Ford
Chair

Dr. John M. Colombi
Member

Lt Col Jeremy R. Geiger
Member

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENV-MS-22-M-237

Abstract

The United Stated Department of Defense (DoD) and Air Force (USAF) have placed increased emphasis on the utilization of modern systems engineering (SE) practices within the current and future acquisitions lifecycle. This call is driven by the current rate at which near-peer adversaries such as Russia and China are increasing their defense system capabilities and catching up or surpassing the Unites States in certain operational regions. To aid in the transition to a Digital Engineering and Digital Twin dominated acquisitions process, this thesis presents a method with which SysML and geometric tools can be linked within both new and existing models built through MBSE practices. Specifically, this thesis focuses on the use of Cameo Systems Modeler and Engineering Sketch Pad to explore the link between Systems Models and Geometric Models. These tools, in conjunction with a well-developed pattern were exercised using a simple, ground-up approach model and an existing model. The result of this thesis is a stepping-stone to more complex geometric generation and a direct pipeline from SysML to analysis tools which require either solid models or volume meshes (i.e., CFD, FEM, RCS calculations).

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr Thomas Ford, for his guidance and support throughout the course of this thesis effort. His knowledge and understanding guided this research tremendously. I would, also, like to thank my fiancé, family, and friends who have helped me along my thesis and overall Air Force journey. Without them, none of this was possible.

Alexander J. Miesle

Table of Contents

List of Figures

List of Tables

TITLE

I. Introduction

1.1 General Issue

The United States Department of Defense (DoD) and Air Force (USAF) have placed increased emphasis on the utilization of modern systems engineering (SE) practices within the current and future acquisitions lifecycle. This call is driven by the current rate at which near-peer adversaries such as Russia and China are increasing their defense system capabilities and catching up or surpassing the Unites States in certain operational regions (Defense, 2018). One of the modern SE practices which has seen increased focus is Digital Engineering, and more specifically the area of Model Based Systems Engineering, wherein a "Digital Twin" of a system is created which can be utilized to explore modifications, support testing, and track impact of changes on cost, schedule, and performance (Roper, 2020).

This process starts as early as the initial analysis of alternatives for a system, wherein multiple different system designs are compared against their ability to perform the desired system mission. This period of the Acquisitions Lifecycle, often described as "Conceptual Design," provides freedom of choice and change at relatively cheap cost to the overall system lifecycle, shown in Figure 1, and is arguably the most important step in ensuring the right system is chosen to provide warfighters with their needed capabilities (Tarkian, 2009). As the DoD and USAF strive to incorporate these modern SE processes earlier and more often in the Acquisitions Lifecycle, emphasis must be placed on increasing the methods available to systems engineers to analyze and convey their system design and capabilities.

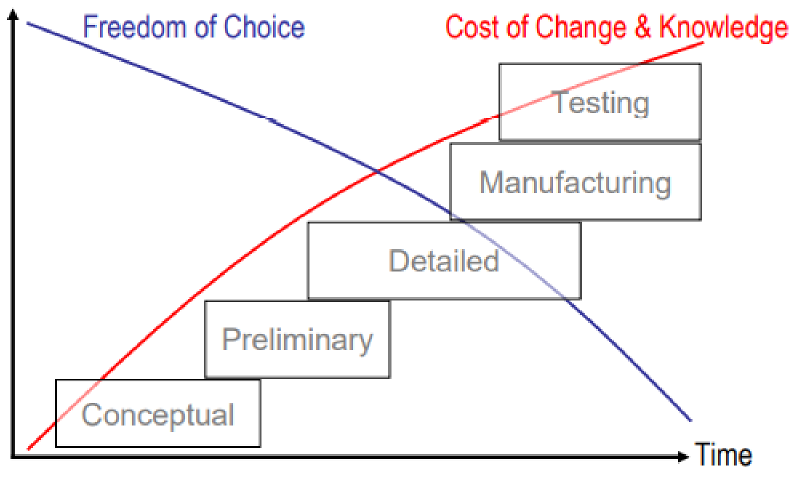


Figure 1: Freedom of Choice vs. Cost of Change and Knowledge (Tarkian, 2009)

1.2 Problem Statement

There is a stove-piped gap between SysML based systems modeling and parametrically defined geometric modeling of a system. Currently, there is little evidence Conceptual Design within Systems Engineering tools which utilize Systems Modeling Language (SysML) have a robust way of providing a geometric representation of a system directly from the defined systems model. Often, the first-time geometric design is completed is at the Detailed Design levels utilizing Computer Aided Design (CAD) tools focused on manufacturing CAD (mCAD) as opposed to analysis CAD (aCAD).

1.3 Research Objectives/Questions

The objective of this research is to eliminate the stove-piped gap between SysML systems modeling and parametrically defined geometric modeling of a system by 1) integrating an MBSE tool with a Geometric modeling tool and 2) defining a method for integrating the system model for each. This objective can be reached by answering the following questions:

1. What difficulties are present in integrating analysis-centric geometry generation with commonly used SysML MBSE tools? Where does the current SysML language fall short?
2. How does the SysML model need to change for a geometric interface to be created within existing SysML MBSE model frameworks without impacting system design?
3. What pattern should be followed to integrate the SysML model and the geometric model? Can this be done in a decoupled fashion so as to eliminate (or minimize) the changes to the original SysML model?
4. How can SysML and geometric modeling integration be demonstrated using a simple example? Highlight the pattern, benefits, and challenges of the approach.

1.4 Methodology

To ensure the methods developed in this effort do not become burdensome on systems engineers and acquisitions professionals, this research will be focused on designing a process or pattern within the bounds of SysML, which can be utilized in addition to current SE practices to supplement current and future system modeling efforts.

1.5 Assumptions/Limitations

- Use of Cameo Systems Modeler from No Magic as the SysML tool of choice

- Limited to SysML 1.6
- Use of Engineering Sketch Pad V1.20 from Massachusetts Institute of Technology and Syracuse University
- Use of Python as the computer language of choice for any external interface between SysML and ESP
- All data generated to be UNCLASSIFIED and Distribution A

II. Literature Review

2.1 Introductory Literature

To fully grasp the intention of this thesis, a baseline understanding of the topics to be discussed and utilized must be provided. Three major topics have been identified as necessary to ensure the reader understands the intent and guiding principles for the implementation of the methods and results described in Chapters 3 and 4. These topics are: Systems Engineering with a focus on SysML, Department of Defense Acquisitions Lifecycle, and Conceptual Design.

2.2 What is Systems Engineering?

Systems Engineering is a process which aims to deliver a product (system) from "lust to dust" or concept inception to product obsolescence (Buede, 2016). According to an IEEE Journal Entry in 2000, Systems Engineering *"involves conceptualization, design, development, test, implementation, approval/certification and operation (including human factors) of a system* (IEEE, 2000).*"* Unlike discipline-based engineering such as Mechanical, Electrical, Aerospace, Civil, etc., Systems Engineering strives to understand the interconnectedness of these disciplines within a system, as well as non-engineering disciplines such as Logistics, Operations and Maintenance, and

Airworthiness Assessments. The desire of this thesis is to enable more comprehensive practices during the conceptualization phase of System Engineering through the addition of geometric modeling within the Systems Modeling Language (SysML). To explore how geometric design can be done using Systems Engineering, it is ideal to explore the different functions of the design process shown in Table 1, as outlined by Dennis Buede in *The Engineering Design of Systems* (Buede, 2016). While these functions look to be logically aligned in a waterfall type of approach, they are, in practice, completed in a parallel manner where each function has inputs and outputs which are dependent on other functions in the design process (Buede, 2016).

Table 1: Systems Engineering Design Functions (Buede, 2016)

Function
0a: Define the problem to be solved
0b: Define and evaluate alternate concepts for solving problem
1: Define the system level design problem being solved
2: Develop the system functional architecture
3: Develop the system physical architecture
4: Develop the system allocated architecture
5: Develop the interface architecture
6: Define the qualification system for the system

Considering phase 0a and 0b, often referred to as Concept Initiation, geometric designs can play a vital role in ensuring the concept elements which are being described have validity. In Department of Defense (DoD) Acquisitions, the problem to be solved is often condensed down into a capability requirement. There is danger here in that a complex solution which looks to be the most efficient option to achieve the desired effect may not be the right one. There is a global System of Systems problem that must be considered at this stage to ensure the proposed concept is feasible within the domain of operations which this system will exist (Gillespie, 2017). For example, when developing an air-to-ground missile (AGM), if the system design does not include a geometric model, there could be overlooked issues such as volumetric packaging or operationally limiting factors such as vehicle footprint on a runway or in a hangar. This need for validation through geometry is addressed specifically to Mechatronics in a paper by Aude Warniez, wherein the need for reducing physical impacts of a multi-physical system is critical (Warniez, 2017).

2.3 Conceptual Design

When moving out of Concept Initiation and into the 6 primary functions in the design process, geometric design becomes much more involved. We are no longer only looking externally, but internally at the system and how its components interact with each other. The power of MBSE allows the individual geometries of each part to be defined. As these geometries are defined, one can identify which components do not interface with each other, and where a new component design must be selected. Performing this task early and often is important to ensuring the system does not fail during integration.

In the interest of performing geometric design early, it is ideal to start at the beginning of a system's lifecycle. This phase, which includes the Concept Initiation as described above, will be referred to as Conceptual Design. Currently, there are many discussions about how to perform this Conceptual Design in an MBSE environment, specifically called Model Based Conceptual Design (MBCD). The main concept behind MBCD is to reduce the number of requirements on a system using a Goals-Needs construct (Hummel, 2016). The idea is that requirements put too many constraints on system design and restrict the freedom of the engineer to arrive at an innovative solution in lieu of "checking the box" on the listed requirements. If one uses the method of classifying needs, then goals can be used to satisfy that need. If the system design within the MBSE tool does not achieve the needs of the customer, then it is not a valid solution. From the perspective of Research and Development (R&D), this idea is extremely valuable. Using a Goals-Needs mentality would be extremely useful in early tech development, where the scope of the problems being approached is much wider and less defined than a typical acquisitions program. This approach, paired with doing early geometric design, would allow the systems engineer to create models early in the design process. This would enable trade space analysis and Analysis of Alternatives (AoA) against a baseline design, to determine the best possible option as early as possible. The level of acceptance for these designs will be in question and requires careful attention to how detailed the geometries and the overall system are modeled. A common saying in modeling is that "all models are wrong; some are useful" – George E.P. Box. To best convey the "level of wrongness," the terms of abstraction and fidelity level must be understood and properly tracked. Abstraction can be considered how far from reality a system is represented, and fidelity level is the level of "exactness" of the model or simulation relative to the real-world environment (Hunter, 2015). If each geometric design does not reflect the same level of abstraction as the rest of the model, then the accuracy of the model is brought into question. Therefore, while early geometric design is important, if done incorrectly it can become more of a hindrance than a help in system design.

As systems mature and move through their lifecycle, delays, changes, modifications, and cost overruns begin to become apparent. As shown in Figure 2 below, early in a design the cost to make changes and try different configurations is low, but as the system becomes more concrete and detailed, the cost of those changes begins to climb (Tarkian, 2009).

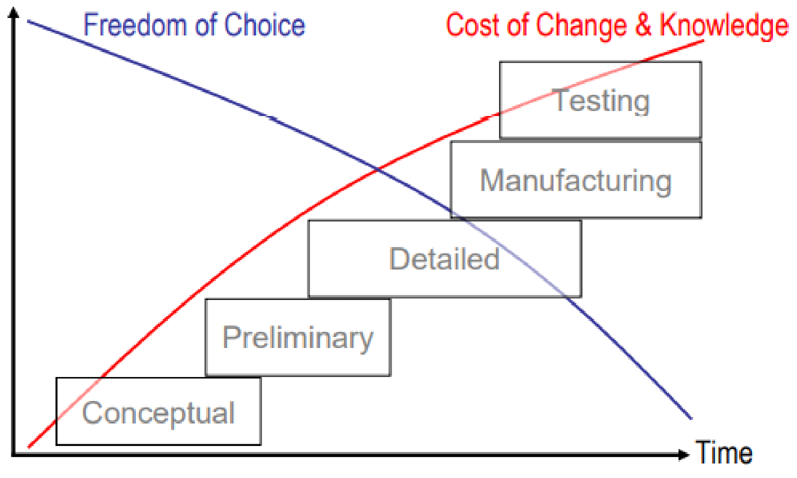


Figure 2: Freedom of Choice vs. Cost of Change and Knowledge (Tarkian, 2009)

Using MBSE early in the process to help define the system from Goals and Needs to the Functional Architecture and the various potential Physical Architectures which could exist is extremely beneficial. The more knowledge gained early in the design process allows the systems engineer to make a more educated and capability driven decision on what component should be selected for the system.

Conceptual modeling can be powerful *if applied correctly*. This means that the model fidelity must not exceed the level of detail needed to solve the problem the model was designed for. As models become more and more complex, there is a risk that the compounding model uncertainties will eventually render the model useless. In practice, this will be difficult to implement, as the desire to quickly field a system often requires the use of COTS and historical information to quickly skip to detailed design phases. A great analogy used by Thomas Lucas on model complexity is the following poem (Lucas, 2002):

For want of a nail the shoe was lost

For want of a shoe the horse was lost

For want of a horse the rider was lost

For want of a rider the message was lost

For want of a message the battle was lost

For want of a battle the kingdom was lost

And all for the want of a nail

Lucas follows this poem with a conversation on how it is impossible to truly model the complexities of reality, and that the designer (or analyst in his paper) must know how to apply the right level of fidelity (Lucas, 2002).

2.4 Department of Defense Acquisitions

2.4.1 Digital Engineering

Acquisition of new materiel in the United States Department of Defense (DoD), and more specifically in the US Air Force (USAF) is a complex process, currently regulated through a series of required documentation and dictated events which describe how to acquire new or upgraded technologies and equipment. This process is often bogged down by specifics regarding these documents and processes, which are often interpreted as detractors rather than productive actions surrounding the system design and acquisition. Currently, the DoD is striving to create a shift in the way this acquisition is performed through the implementation of a Digital Engineering Strategy. This strategy is focused on “the challenge of balancing design, delivery, and sustainment of complex systems with rapidly changing operational and threat environments” (ODASD(SE), 2018). The emphasis of this document is not on one specific way of performing Digital Engineering, but rather to describe a shift from document-based processes to utilizing a continuous model which exists throughout the system lifecycle and provides the information required for decision making. The goal of managing systems in this way is to foster rapid and responsive development environments which will aid in addressing the ever-increasing complexity and rate of change in the military environment (ODASD(SE), 2018). The DoD Digital Engineering Strategy outlines five goals which are meant to aid in the process of integrating model-based digital engineering into the current DoD Acquisitions process, outlined in Table 1.

Table 2: Digital Engineering Goals (ODASD(SE), 2018)

Digital Engineering Goals
1. Formalize the development, integration, and use of the models to inform enterprise and program decision making
2. Provide an enduring, authoritative source of truth
3. Incorporate technological innovation to improve the engineering practice
4. Establish a supporting infrastructure and environments to perform activities, collaborate, and communicate across stakeholders
5. Transform the culture and workforce to adopt and support digital engineering across the lifecycle

These 5 goals illustrate the desire of the DoD to pursue a Digital Engineering framework which enables communication up and down the authoritative chain, between personnel with varying levels of experience and expertise, and from system inception to system end-life. While Systems Engineering is capable of describing an entire system through

SysML and other tools, it is often difficult for everyone who is interacting with the model to arrive at the same physical description in their head. Similar to reading a book, people often have different experiences in how they process the information being provided to them. Enabling a geometric interface within SysML would drive towards a much easier to understand system, where text-based definitions within a SysML tool are directly attributed to a physical model for everyone to see and manipulate. This type of interaction could significantly aid in Goals 4 and 5, which focus on the ability to communicate and adapt to the adoption of Digital Engineering in the DoD.

2.4.2 DoD Acquisitions Lifecycle

To aid in understanding the expected implementation of this research, a basic understanding of the DoD Acquisitions Lifecycle must be developed. Figure 3 illustrates the general lifecycle for physical materiel acquisition in the DoD (DAU, 2013).

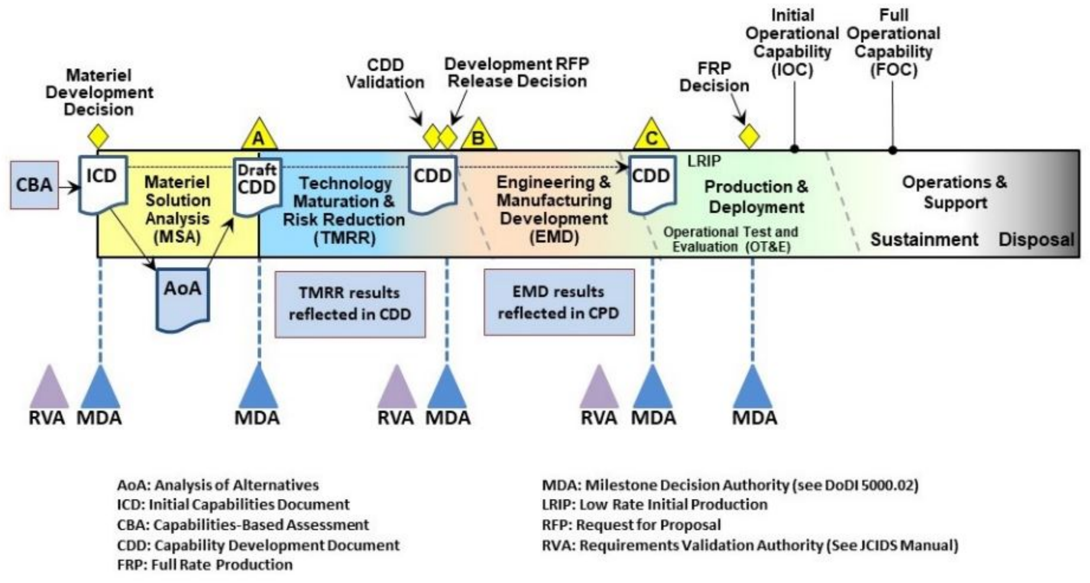

Materiel Development Decision
CDD Validation
Development RFP Release Decision
FRP Decision
Initial Operational Capability (IOC)
Full Operational Capability (FOC)
A
B
C
CBA
ICD
Materiel Solution Analysis (MSA)
Draft CDD
Technology Maturation & Risk Reduction (TMRR)
CDD
Engineering & Manufacturing Development (EMD)
CDD
LRIP
Production & Deployment
Operational Test and Evaluation (OT&E)
Operations & Support
Sustainment
Disposal
AoA
TMRR results reflected in CDD
EMD results reflected in CPD
RVA
MDA
MDA
RVA
MDA
RVA
MDA
MDA
AoA: Analysis of Alternatives
ICD: Initial Capabilities Document
CBA: Capabilities-Based Assessment
CDD: Capability Development Document
FRP: Full Rate Production
MDA: Milestone Decision Authority (see DoDI 5000.02)
LRIP: Low Rate Initial Production
RFP: Request for Proposal
RVA: Requirements Validation Authority (See JCIDS Manual)

Figure 3: Department of Defense Acquisitions Lifecycle

The intent of this thesis is to lay the groundwork for a process which can effectively be implemented during the Pre-Milestone A phase up to Milestone B. Further development will be needed for the specific methods approached here to be valid past TMRR.

2.5 SysML

2.5.1 What is SysML?

Systems Modeling Language (SysML) is a modification of Unified Modeling Language (UML) which is used for software design and engineering. SysML is a language which consolidates numerous elements of definition (akin to classes in programming) and elements of execution (akin to objects in programming) (OMG, 2022). These elements enable a modeler to develop a complex system definition within a digital framework where not only the system is described, but its external and internal component interactions, capability analyses can be performed, and version control can be implemented.

2.5.2 Merging of SysML and Geometric Design

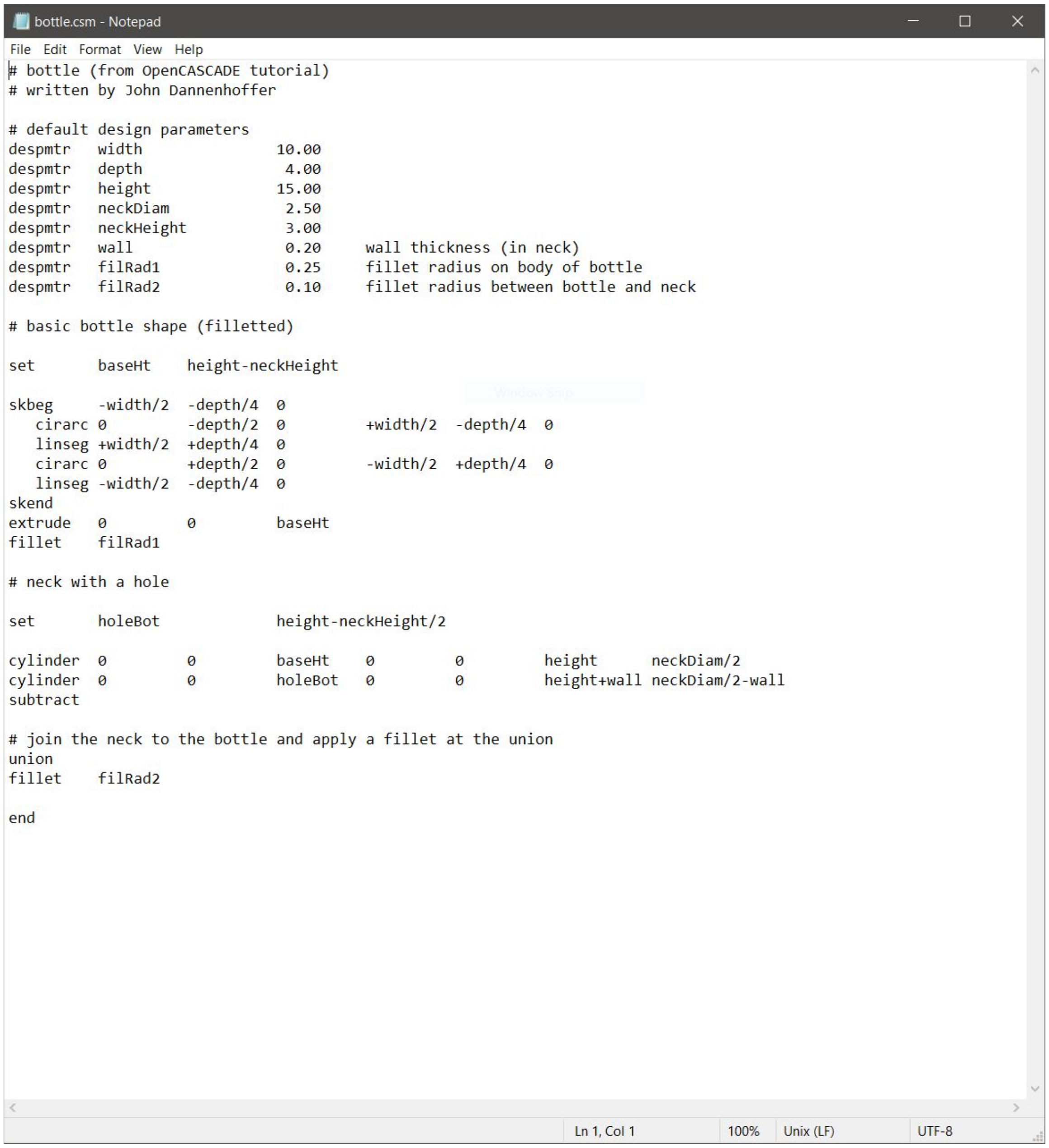
To address the problem statement of this thesis, the overall effort must strive to truly merge the capabilities held within SysML and Engineering Sketch Pad (ESP). The capabilities which should be held important are 1) Systems modeling capabilities held within SysML and 2) The geometric description and analysis capabilities of ESP. These capabilities when merged together can create a powerful end state, wherein system description is owned and managed within a SysML. Through ESP more complex analysis tools not traditionally accessible through SysML such as Cart3D and Fun3D (computational fluid dynamics tools), Sentry (thermal analysis tool from SpaceWorks

Enterprises, Inc.), and Abaqus (structural analysis tool) can be accessed and utilized (Haimes, 2013). Careful consideration must be done on the methods and processes used to address this interaction between SysML and ESP. If done incorrectly, either the power of SysML or ESP will be lost. One method previously explored is to create a Geometric Profile, wherein specific geometries are defined through the use of stereotypes. These stereotypes are a simplified representation of a physical component in the system, limited to ~10 simple shapes. For example, a nose cone with complex geometric fairings would realistically be simplified down to a basic cone (Warniez, 2017). Naturally, there will be some losses associated with merging the two tools in a preliminary way. The focus of this thesis will be to maintain as much access to SysML capabilities as possible, therefore, some ESP functionality may be lost in the process.

2.6 Engineering Sketch Pad

2.6.1 What is Engineering Sketch Pad

Engineering Sketch Pad (ESP) is a geometric modeling tool which focuses on enabling interfaces between geometric modeling and analysis tools. This is accomplished through the use of Open-Source Constructive Solid Modeler (OpenCSM) which provides access to both a feature tree and design parameters simultaneously (Haimes, 2013). ESP differentiates itself from most modern CAD programs (Catia, SolidWorks, UniGraphics, etc.) using open-source software and the user interface through which the modeler engages with the tool. Traditionally, CAD programs have "primitives" (i.e., circles, squares, boxes, splines, etc.) which are used in conjunction with one another to generate a geometric representation of a system. These are most often GUI-based interactions, which are defined in the software and cannot be modified to meet a modeler's specific needs (Haimes, 2013). In ESP, the modeler designs through the use of a scripting language specific to the OpenCSM framework. An example of the scripting language layout can be seen in Figure 4.



```
# bottle (from OpenCASCADE tutorial)
# written by John Dannenhoffer

# default design parameters
despmtr   width               10.00
despmtr   depth                4.00
despmtr   height              15.00
despmtr   neckDiam             2.50
despmtr   neckHeight           3.00
despmtr   wall                 0.20     wall thickness (in neck)
despmtr   filRad1              0.25     fillet radius on body of bottle
despmtr   filRad2              0.10     fillet radius between bottle and neck

# basic bottle shape (filletted)

set       baseHt    height-neckHeight

skbeg     -width/2  -depth/4  0
   cirarc 0         -depth/2  0         +width/2  -depth/4  0
   linseg +width/2  +depth/4  0
   cirarc 0         +depth/2  0         -width/2  +depth/4  0
   linseg -width/2  -depth/4  0
skend
extrude   0         0         baseHt
fillet    filRad1

# neck with a hole

set       holeBot             height-neckHeight/2

cylinder  0         0         baseHt    0         0         height      neckDiam/2
cylinder  0         0         holeBot   0         0         height+wall neckDiam/2-wall
subtract

# join the neck to the bottle and apply a fillet at the union
union
fillet    filRad2

end
```

Figure 4: Example ESP Script from bottle.csm Provided in ESP Package

This script-based interaction allows for common programs such as Python, C+, and Java to interface directly with the inputs for the geometric designs. Specifically, a Python-based interface named pyCAPS has been generated to enable access to many of the capabilities of ESP from a user-focused lens, striving to bypass the need for compiling of C-based code and utilizing the nature of Python as an "easy to learn, readable (compared to low-level programming languages)" language (Durscher, 2019). While pyCAPS is very useful, it is not capable of a quick interface with SysML, as it is a JavaScript based environment. Along with the script-based implementation of geometric design, ESP utilizes the visualization capabilities of modern web browsers to display the designed object and its information in an easy format, but does not bypass the need for the data to be available to the modeler. Through ESP, the modeler can 1) modify the implementation of primitives in the model, 2) create new user defined primitives (UDPs) which are akin to functions in traditional programming, and 3) interface with the tool through a text-based file written in the OpenCSM syntax (Haimes, 2013).

2.6.2 How is ESP Useful in Systems Engineering?

The usefulness of ESP in Systems Engineering comes from the flexibility of the tool to be interfaced with externally, which allows for the 3D conceptualization of the System Design for Concept evaluation and analysis. ESP traditionally is designed to be the geometric "data manager" for designs. A modeler generates a design, which can be executed on through OpenCSM, and then the data relative to that design can be passed to multiple different analysis tools simultaneously. So how do Systems Engineers take advantage of this tool to improve their practices? Due to the ease of integration of ESP relative to other geometric software programs, it can be postulated the not only can a path from ESP to another tool be utilized, but also a path from another tool to ESP. Essentially, instead of generating the feature tree directly in ESP, we now utilize SysML as the owner of the system configuration and use the information stored there to generate a valid OpenCSM file to be executed. This path is illustrated in Figure 5 and Figure 6.

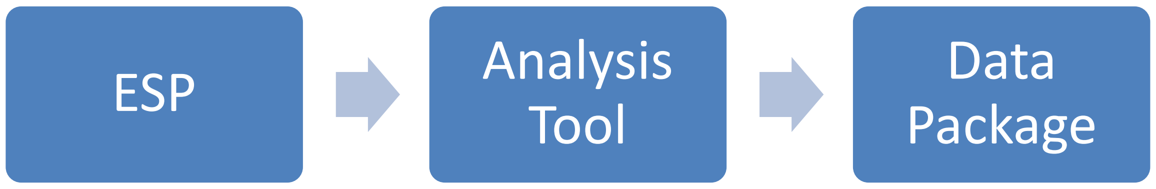


Figure 5: ESP Data Path

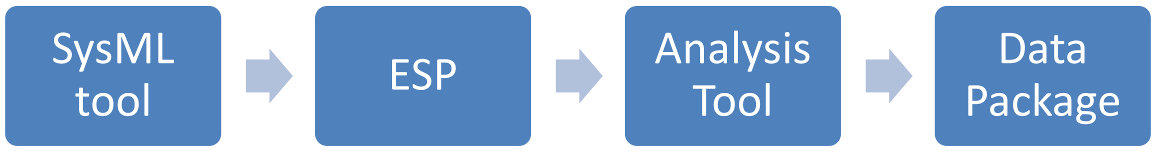


Figure 6: ESP + SysML Data Path

The overall usefulness of this pathway comes from the ability of SysML to define a system which can be executed and analyzed. Once the system design is developed in SysML, information such as part-whole relationships and system instances can be extracted and acted upon through ESP.

2.6.3 Example of ESP Implementation

As an example of how ESP has been used in modeling efforts, we will explore a paper by Justin Clough titled *Automated Wing Internal Structure Placement Guided by Finite Element Analysis*. This paper explores an iterative process through which a model is defined, analyzed, and modified to meet a set of requirements. The methods used in this paper regarding structural design of a wing and analysis methods are outside the scope of this thesis, but the overall process executed can be abstracted to develop a methodology useful to systems engineering. Clough identifies a problem initiation point wherein a geometry which can be manipulated is generated (Clough, 2019). This geometry is then able to be analyzed through Finite Element processes and the information fed back to inform the next iteration of the design. The geometry initiation point in Clough's research was performed directly in ESP and the analysis-based decisions were managed with a Python script. Both of these actions could be directly translated to a SysML tool and integrated with ESP, almost like a wrapper where ESP's capabilities are directly accessed by SysML. Not only was Clough able to prove through this research that ESP is capable of receiving new information and acting on it but is able to work in conjunction with non-geometric properties such as material type, density, and strength as well as a pseudo-requirement of driving a design down towards minimum tip deflection for an example wing (Clough, 2019). These two concepts in Clough's work can be traced to Value Properties of Blocks and Requirements in SysML, showing that the use of ESP could fit within the current Systems Engineering and SysML framework without requiring extensive changes to how SysML is utilized.

III. Methodology

3.1 Research Objective

3.1.1 Overview: Stovepiped Gap Between SysML and Geometric Modeling

For this thesis, it is paramount to identify the objective, the research questions, and the overall method used to develop a pattern which enables passing of geometric data between a SysML model and a parametrically defined geometric description of the system. There is a stovepiped gap between SysML and Geometric Modeling, and in order to eliminate this gap, we must integrate an MBSE tool with a geometric modeling tool and define a method which can be replicated to integrate the System Model with the Geometric Model. A basic description of this method is described in Figure 7.

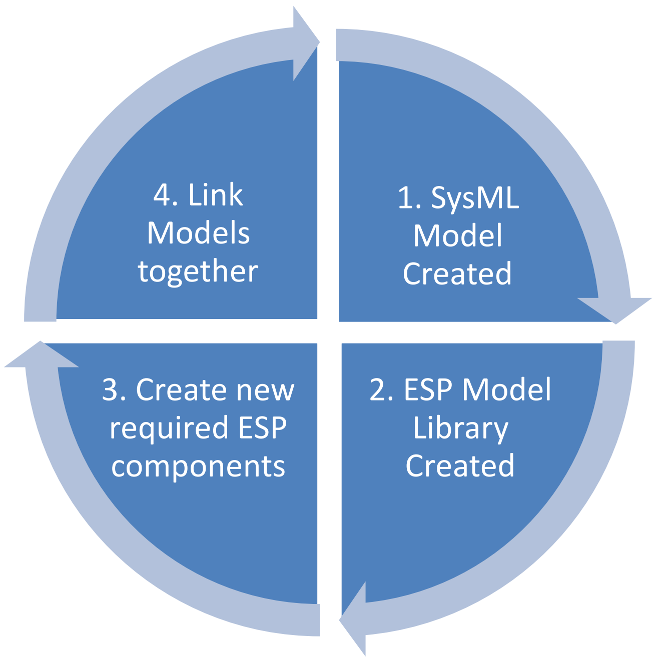

1. SysML Model Created
2. ESP Model Library Created
3. Create new required ESP components
4. Link Models together

Figure 7: Basic SysML to Geometric Tool Process

The stovepiped gap which exists currently is an artifact of the use of individual tools on the part of both the geometric modeler and the systems modeler, wherein these tools lack a link which allows for passing of data easily between two differently defined representations of the same system. This also leads to separate and asynchronous data sources caused by a lack of a unifying model ownership, namely within SysML and MBSE tools.

3.1.2 Tool Integration

3.1.2.1 Externally Linked Tools

To meet the first step of eliminating this stovepiped gap, the tool integration must be achieved in a way which both system descriptions (System and Geometric) have access and knowledge to the same information. The tool integration options investigated and tested can be defined in two ways: First, externally linked tools; Second, internally linked tools. These are defined respectively by A) the addition of a separate and external method for passing and managing the data handled to define the geometric description of the system or B) utilizing the power of either the geometric or MBSE tool to own the management of the geometric data. For the externally linked tools method, the most viable option is to utilize the data exporting capabilities that are native to Cameo Systems Modeler, namely exporting value properties through a Comma Separated Value (CSV) function using a Simulation Context or exporting the model in an XMI format which can then be parsed through the use of a modern coding languages (i.e., Python, Java, C+) to extract the required data. The data extracted from SysML is then processed utilizing a script which must have prior knowledge of the model and access to the ESP environment.

These methods, while capable of producing a geometric model, do not meet the intent of this thesis as they remove the power of SysML from the overall design process and simply extract a current model state to explore the variables and build a geometric description. In conjunction with removing SysML from the process, updates are not automatic and if geometric modeling happens at a pace slower than the overall systems modeling effort, there will be versioning issues and an inherent trail-behind of geometric description relative to the current system description. This also removes the capability that system provides with regards to Monte Carlo simulation and Trade Studies to rapidly assess variations in the system model and provide insight into the overall System capability.

3.1.2.2 Internally Linked Tools

Regarding internally linked tools, the data that is required and the pattern with which this data is handled does not necessarily change. Rather the ownership of the data management is moved to within a SysML tool or can utilize ESP functionality through CAPS or pyCAPS. The DoD efforts of a Digital Engineering transformation lead the focus of this portion of the research to naturally prioritize MBSE or SysML tools to be the owner of data management and the source of truth over ESP. This method was found to require one external process to be completed prior to modeling, wherein premade text files, which can be updated by SysML through Opaque Behaviors, are placed in an existing or known location on the computer which is executing the model. These external efforts are meant to enable the power of SysML for future work regarding trade space studies and potential API integration with other existing tools. This also eliminates reliance on manual updates of external data packages, which inherently comes with the externally linked tool methods described previously.

3.1.3 Chosen Method for Tool Integration

Because the desire of this thesis is to investigate how to merge systems modeling practices with geometric modeling, the internally linked tools method was quickly identified as the preferred solution over externally linking the two tools with a separate process. This is due to the fact that any attempts to externally link SysML and ESP lead to the loss of nearly all SysML capabilities. To achieve this, an ESP Model Library and a Systems Model were included in a single project. The purpose of the ESP Model library is to build Activities and Opaque Behaviors which are common to enabling the Cameo to ESP interface. Also, common geometric shapes (such as spheres, cylinders, boxes, etc.) can be stored in this Model Library for future use in the current or existing model. The use of a Model Library allows for decoupling of the SysML Model from the ESP Model being developed. This means the System Model can exists prior to the Geometric Model and is relatively unchanged when the ESP Model Library is added. It was discovered this process requires intelligent modeling of relationships between the System and ESP models which allow the retention of ownership of System-related properties within the Systems Model and Geometric-related properties to be owned and handled in the ESP model. This relationship is achieved through an association which is bilateral in nature as opposed to commonly used unilateral associations (i.e., Straight Line Association vs Directed Association). The difference between these two associations can be seen in Figure 8 and Figure 9.

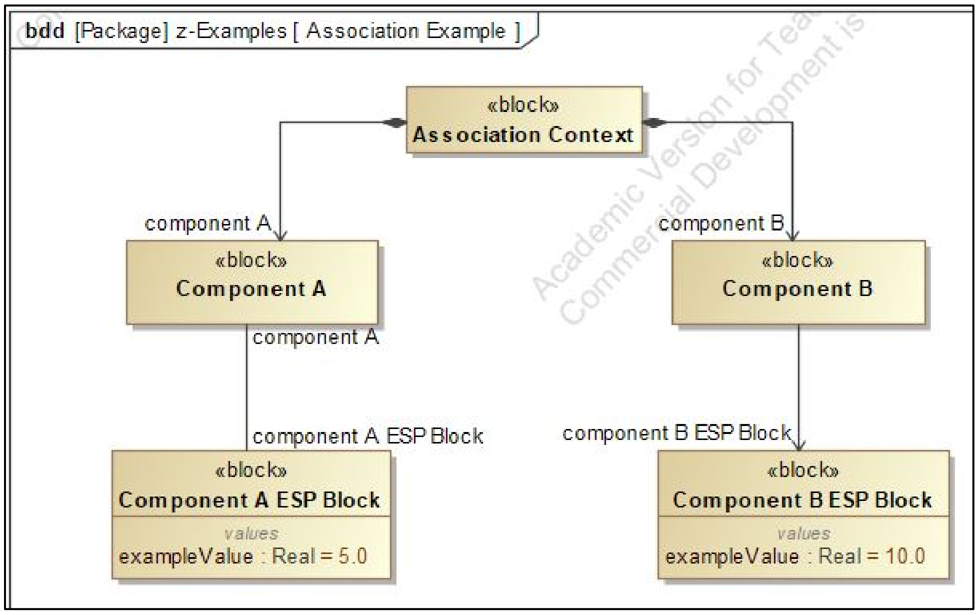
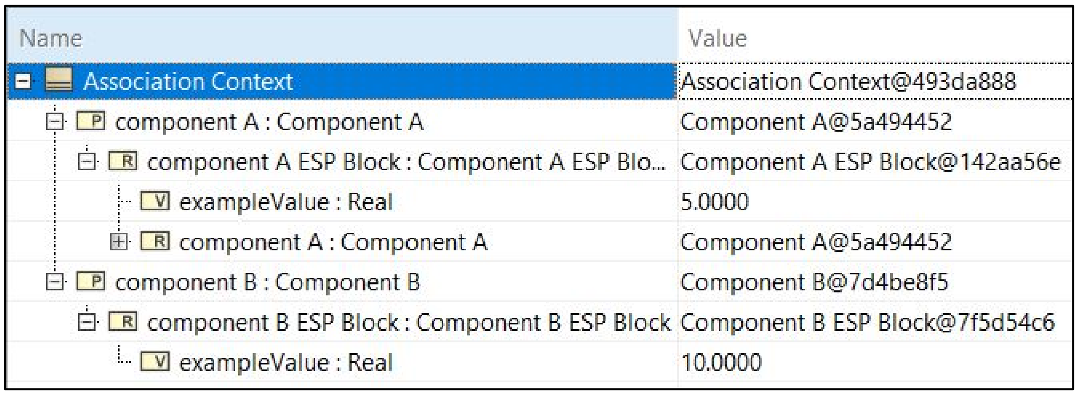


Figure 8: Association Context Example



Name	Value
Association Context	Association Context@493da888
component A : Component A	Component A@5a494452
component A ESP Block : Component A ESP Blo...	Component A ESP Block@142aa56e
exampleValue : Real	5.0000
component A : Component A	Component A@5a494452
component B : Component B	Component B@7d4be8f5
component B ESP Block : Component B ESP Block	Component B ESP Block@7f5d54c6
exampleValue : Real	10.0000

Figure 9: Association Context Simulation Example

Figure 8 shows a Block Definition Diagram containing a Context Block and two Components A and B of which the Association Context is directly composed. Component A and B are both Associated to an "ESP Block" through the two different Association types mentioned. Figure 9 is the result of simulating the Association Context in Figure 8, and shows how the Straight-Line Association creates a recursive relationship between

Component A and Component A ESP Block, where each is a reference property of the other. Figure 9 also shows how the Directed Association creates an instance of Component B which contains Component B ESP Block as a Reference Property but does not have Component B as a Reference Property of Component B ESP Block.
The potential issues caused by the circular reference created with a straight-line association were deemed to be negligible in terms of the research being performed for this thesis but may pose complications if automation of larger systems in the future is not done carefully. This also increases the manual processes required to fully simulate the Geometric Model, where an "Add Value" step must be done during simulation to properly update Reference Property values in the ESP model with existing values in the System Model. It was identified that large systems with multiple components could potentially make this a tedious task and mitigate the overall benefit of linking the two tools, but it should be noted that the intention of this specific geometric integration is for early concept design and analysis which inherently does not have a large order of magnitude of components.

3.2 Research Questions

To address the research objective for this thesis, 4 Research Questions were identified:

1. What difficulties are present in integrating analysis-centric geometry generation with commonly used SysML MBSE tools? Where does current SysML language fall short?

2. How does the SysML model need to change for a geometric interface to be created within existing SysML MBSE model frameworks without impacting system design?

3. What pattern should be followed to integrate the SysML model and the geometric model? Can this be done in a decoupled fashion so as to eliminate (or minimize) the changes to the original SysML model.

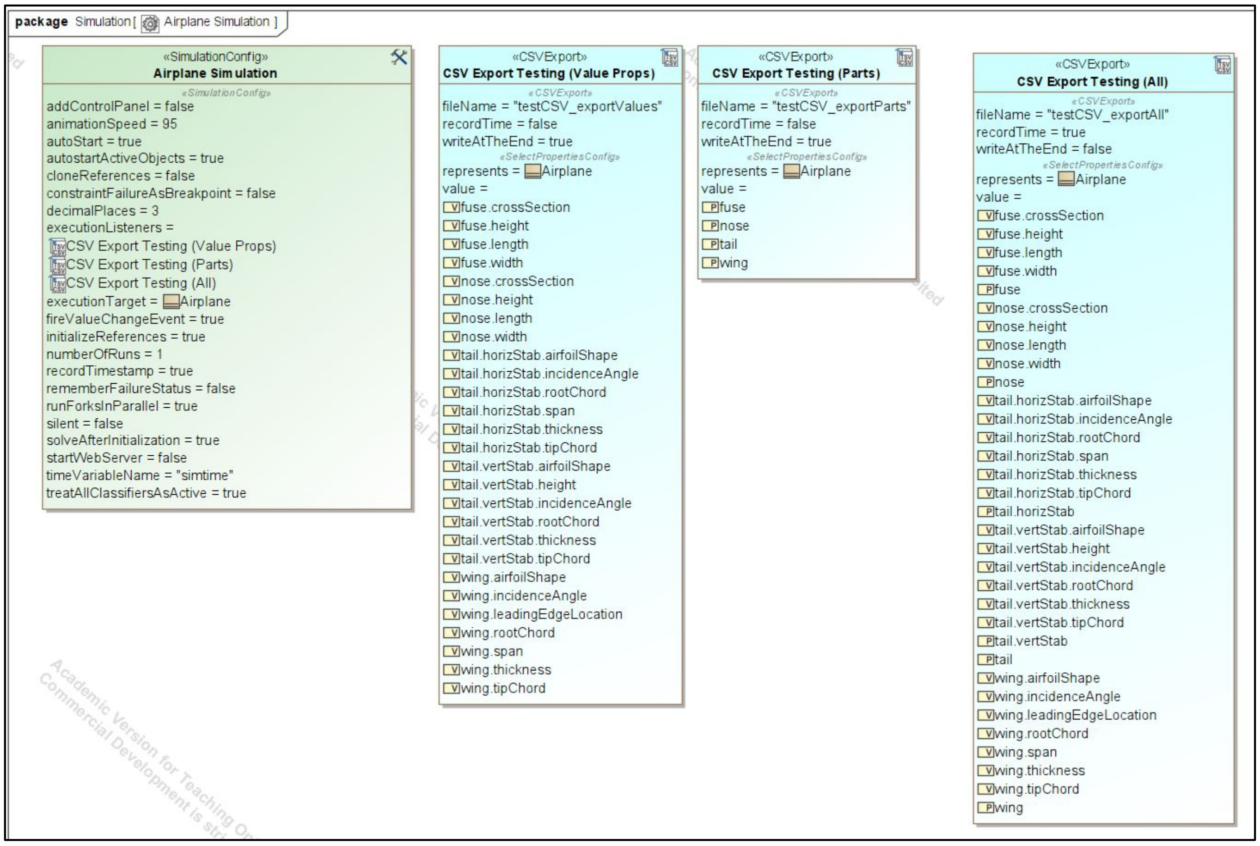
4. Demonstrate the SysML and geometric modeling integration using a simple example. Highlight the pattern, benefits, and challenges of the approach.

3.2.1 Research Question 1

The first research question focuses on identifying the difficulties with integrating analysis centric geometry with common SysML tools used and strives to identify where SysML may fall short in enabling geometric tools to easily interface with the data available. The major difficulties identified were 1) data exporting typically removes the capabilities inherent to SysML and 2) the model must be made in such a way that the data can be pulled in a logical way.

3.2.1.1 CSV Export Method

The first option considered was the CSV export which comes native to Cameo Systems Modeler.



«SimulationConfig»
Airplane Simulation
«SimulationConfig»
addControlPanel = false
animationSpeed = 95
autoStart = true
autostartActiveObjects = true
cloneReferences = false
constraintFailureAsBreakpoint = false
decimalPlaces = 3
executionListeners =
CSV Export Testing (Value Props)
CSV Export Testing (Parts)
CSV Export Testing (All)
executionTarget = Airplane
fireValueChangeEvent = true
initializeReferences = true
numberOfRuns = 1
recordTimestamp = true
rememberFailureStatus = false
runForksInParallel = true
silent = false
solveAfterInitialization = true
startWebServer = false
timeVariableName = "simtime"
treatAllClassifiersAsActive = true

«CSVExport»
CSV Export Testing (Value Props)
«CSVExport»
fileName = "testCSV_exportValues"
recordTime = false
writeAtTheEnd = true
«SelectPropertiesConfig»
represents = Airplane
value =
V fuse.crossSection
V fuse.height
V fuse.length
V fuse.width
V nose.crossSection
V nose.height
V nose.length
V nose.width
V tail.horizStab.airfoilShape
V tail.horizStab.incidenceAngle
V tail.horizStab.rootChord
V tail.horizStab.span
V tail.horizStab.thickness
V tail.horizStab.tipChord
V tail.vertStab.airfoilShape
V tail.vertStab.height
V tail.vertStab.incidenceAngle
V tail.vertStab.rootChord
V tail.vertStab.thickness
V tail.vertStab.tipChord
V wing.airfoilShape
V wing.incidenceAngle
V wing.leadingEdgeLocation
V wing.rootChord
V wing.span
V wing.thickness
V wing.tipChord

«CSVExport»
CSV Export Testing (Parts)
«CSVExport»
fileName = "testCSV_exportParts"
recordTime = false
writeAtTheEnd = true
«SelectPropertiesConfig»
represents = Airplane
value =
P fuse
P nose
P tail
P wing

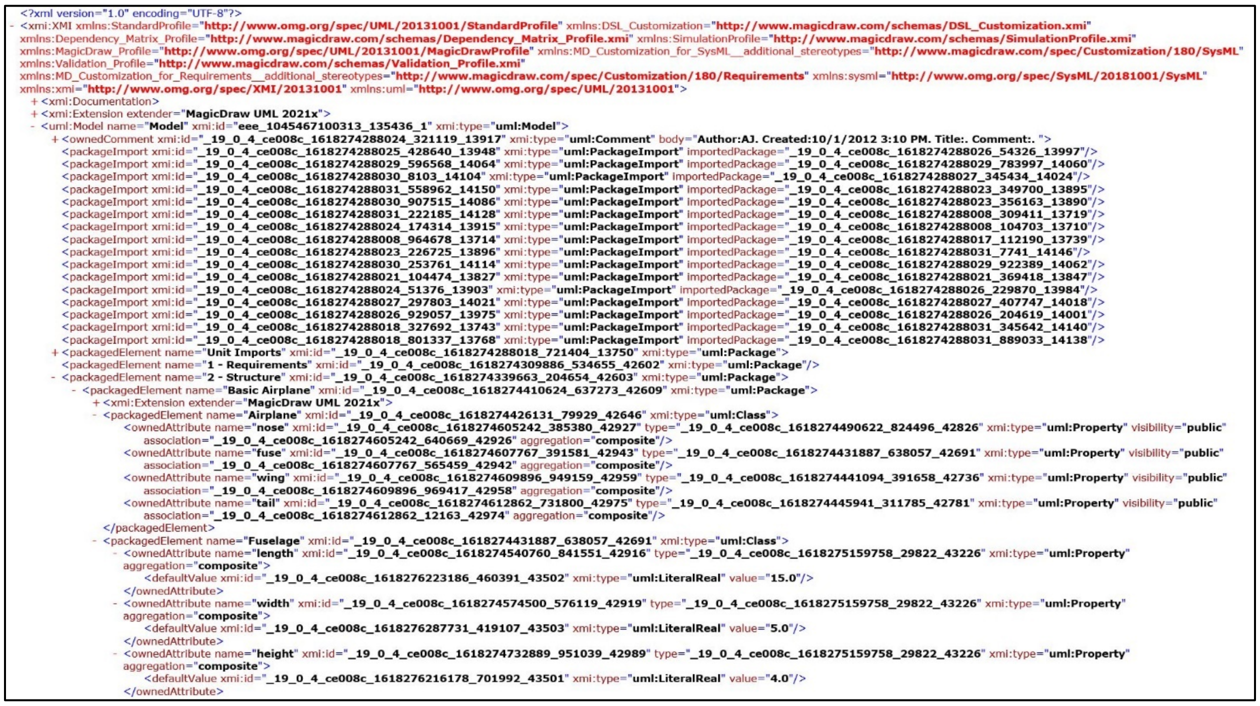
«CSVExport»
CSV Export Testing (All)
«CSVExport»
fileName = "testCSV_exportAll"
recordTime = true
writeAtTheEnd = false
«SelectPropertiesConfig»
represents = Airplane
value =
V fuse.crossSection
V fuse.height
V fuse.length
V fuse.width
P fuse
V nose.crossSection
V nose.height
V nose.length
V nose.width
P nose
V tail.horizStab.airfoilShape
V tail.horizStab.incidenceAngle
V tail.horizStab.rootChord
V tail.horizStab.span
V tail.horizStab.thickness
V tail.horizStab.tipChord
P tail.horizStab
V tail.vertStab.airfoilShape
V tail.vertStab.height
V tail.vertStab.incidenceAngle
V tail.vertStab.rootChord
V tail.vertStab.thickness
V tail.vertStab.tipChord
P tail.vertStab
P tail
V wing.airfoilShape
V wing.incidenceAngle
V wing.leadingEdgeLocation
V wing.rootChord
V wing.span
V wing.thickness
V wing.tipChord
P wing

Figure 10: Simulation Context to Enable CSV Export Within Cameo

The challenge with utilizing a CSV export method is that pre-existing knowledge of the data structure is necessary to be able to intelligently design the data management of the geometric model. As seen in Figure 10, this method requires each Value Property to be identified prior to the simulation of the model. This method removes the dependence on SysML as the owner of the truth model for the system which directly contradicts current guidance being passed through the Department of Defense and the Air Force. This also removes the inherent power gained through the utilization of Systems Modeling tools and causes a loss of model data in exchange for the ease of value property access. While CSV is a common data format used in many tools and data management techniques, it is considered a non-ideal solution when truly attempting to integrate systems modeling tools with a geometric modeling tool.

3.2.1.2 XMI Method

The second method explored was XMI file parsing, where an XMI formatted version of the Systems Model is exported from the systems modeling tool and then queried using a Python-based interface. This allows all model information to be retained (i.e., classes, children, parents, etc.). This data, shown in Figure 11, is structured in such a way to allow for easy access of value properties and other model definitions, as long as the structure of the model is known prior to developing the Python interface methods.



```
  <?xml version="1.0" encoding="UTF-8"?>
- <xmi:XMI xmlns:StandardProfile="http://www.omg.org/spec/UML/20131001/StandardProfile" xmlns:DSL_Customization="http://www.magicdraw.com/schemas/DSL_Customization.xmi"
  xmlns:Dependency_Matrix_Profile="http://www.magicdraw.com/schemas/Dependency_Matrix_Profile.xmi" xmlns:SimulationProfile="http://www.magicdraw.com/schemas/SimulationProfile.xmi"
  xmlns:MagicDraw_Profile="http://www.omg.org/spec/UML/20131001/MagicDrawProfile" xmlns:MD_Customization_for_SysML__additional_stereotypes="http://www.magicdraw.com/spec/Customization/180/SysML"
  xmlns:Validation_Profile="http://www.magicdraw.com/schemas/Validation_Profile.xmi"
  xmlns:MD_Customization_for_Requirements__additional_stereotypes="http://www.magicdraw.com/spec/Customization/180/Requirements" xmlns:sysml="http://www.omg.org/spec/SysML/20181001/SysML"
  xmlns:xmi="http://www.omg.org/spec/XMI/20131001" xmlns:uml="http://www.omg.org/spec/UML/20131001">
   + <xmi:Documentation>
   + <xmi:Extension extender="MagicDraw UML 2021x">
   - <uml:Model name="Model" xmi:id="eee_1045467100313_135436_1" xmi:type="uml:Model">
     + <ownedComment xmi:id="_19_0_4_ce008c_1618274288024_321119_13917" xmi:type="uml:Comment" body="Author:AJ. Created:10/1/2012 3:10 PM. Title:. Comment:. ">
       <packageImport xmi:id="_19_0_4_ce008c_1618274288025_428640_13948" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288026_54326_13997"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288029_596568_14064" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288029_783997_14060"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288030_8103_14104" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288027_345434_14024"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288031_558962_14150" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288023_349700_13895"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288030_907515_14086" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288023_356163_13890"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288031_222185_14128" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288008_309411_13719"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288024_174314_13915" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288008_104703_13710"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288008_964678_13714" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288017_112190_13739"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288023_226725_13896" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288031_7741_14146"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288030_253761_14114" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288029_922389_14062"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288021_104474_13827" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288021_369418_13847"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288024_51376_13903" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288026_229870_13984"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288027_297803_14021" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288027_407747_14018"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288026_929057_13975" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288026_204619_14001"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288018_327692_13743" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288031_345642_14140"/>
       <packageImport xmi:id="_19_0_4_ce008c_1618274288018_801337_13768" xmi:type="uml:PackageImport" importedPackage="_19_0_4_ce008c_1618274288031_889033_14138"/>
     + <packagedElement name="Unit Imports" xmi:id="_19_0_4_ce008c_1618274288018_721404_13750" xmi:type="uml:Package">
       <packagedElement name="1 - Requirements" xmi:id="_19_0_4_ce008c_1618274309886_534655_42602" xmi:type="uml:Package"/>
     - <packagedElement name="2 - Structure" xmi:id="_19_0_4_ce008c_1618274339663_204654_42603" xmi:type="uml:Package">
       - <packagedElement name="Basic Airplane" xmi:id="_19_0_4_ce008c_1618274410624_637273_42609" xmi:type="uml:Package">
         + <xmi:Extension extender="MagicDraw UML 2021x">
         - <packagedElement name="Airplane" xmi:id="_19_0_4_ce008c_1618274426131_79929_42646" xmi:type="uml:Class">
             <ownedAttribute name="nose" xmi:id="_19_0_4_ce008c_1618274605242_385380_42927" type="_19_0_4_ce008c_1618274490622_824496_42826" xmi:type="uml:Property" visibility="public"
               association="_19_0_4_ce008c_1618274605242_640669_42926" aggregation="composite"/>
             <ownedAttribute name="fuse" xmi:id="_19_0_4_ce008c_1618274607767_391581_42943" type="_19_0_4_ce008c_1618274431887_638057_42691" xmi:type="uml:Property" visibility="public"
               association="_19_0_4_ce008c_1618274607767_565459_42942" aggregation="composite"/>
             <ownedAttribute name="wing" xmi:id="_19_0_4_ce008c_1618274609896_949159_42959" type="_19_0_4_ce008c_1618274441094_391658_42736" xmi:type="uml:Property" visibility="public"
               association="_19_0_4_ce008c_1618274609896_969417_42958" aggregation="composite"/>
             <ownedAttribute name="tail" xmi:id="_19_0_4_ce008c_1618274612862_731800_42975" type="_19_0_4_ce008c_1618274445941_311785_42781" xmi:type="uml:Property" visibility="public"
               association="_19_0_4_ce008c_1618274612862_12163_42974" aggregation="composite"/>
           </packagedElement>
         - <packagedElement name="Fuselage" xmi:id="_19_0_4_ce008c_1618274431887_638057_42691" xmi:type="uml:Class">
           - <ownedAttribute name="length" xmi:id="_19_0_4_ce008c_1618274540760_841551_42916" type="_19_0_4_ce008c_1618275159758_29822_43226" xmi:type="uml:Property"
             aggregation="composite">
               <defaultValue xmi:id="_19_0_4_ce008c_1618276223186_460391_43502" xmi:type="uml:LiteralReal" value="15.0"/>
             </ownedAttribute>
           - <ownedAttribute name="width" xmi:id="_19_0_4_ce008c_1618274574500_576119_42919" type="_19_0_4_ce008c_1618275159758_29822_43226" xmi:type="uml:Property"
             aggregation="composite">
               <defaultValue xmi:id="_19_0_4_ce008c_1618276287731_419107_43503" xmi:type="uml:LiteralReal" value="5.0"/>
             </ownedAttribute>
           - <ownedAttribute name="height" xmi:id="_19_0_4_ce008c_1618274732889_951039_42989" type="_19_0_4_ce008c_1618275159758_29822_43226" xmi:type="uml:Property"
             aggregation="composite">
               <defaultValue xmi:id="_19_0_4_ce008c_1618276216178_701992_43501" xmi:type="uml:LiteralReal" value="4.0"/>
             </ownedAttribute>
```

Figure 11: XMI File Format Example

Essentially, unless a comprehensive method for parsing all potential data in the XMI file is created, the modeler must know the relative location of all pertinent information prior to creating the parsing script. A potential way around this is through the use of stereotypes, where all ESP-related Blocks and data are typed in the same way. While more comprehensive and less reliant on manual processes, this method removes the capability inherent to systems modeling tools because the model itself is exported into a static state and cannot be operated on any further within the tool. Again, this is a non-ideal solution for integration and is essentially utilizing currently existing data in a separate format to gain access to already existing values.

3.2.1.3 Internal Integration in SysML

The third and final method explored was the internal integration within a Systems Modeling Tool. This enables the capabilities of SysML to be used for future research within systems engineering disciplines, specifically Monte Carlo simulations, trade study analysis, and API integration with separate analysis tools to gain a cohesive understanding of the trade space within which the system will exist. This moves the reliance of the data management over to the existing capabilities of SysML, which may be lacking relative to the abilities of Python, Java, or C+ based practices which are common in current industry endeavors. This also limits the amount of model structure knowledge needed prior to the development of the method to be used, which leads to the chosen approach being an internally linked model which meets the intent of the research and enables future research to be done within a systems engineering scope and not simply developing a Python script which intelligently sifts through extracted data in a common format.

3.2.2 Research Question 2

The second research question focuses on how a systems model would need to change for a geometric interface to be created and how can this be achieved without impacting the system design so that this method can be applied to either a newly developed system or a pre-existing system which does not have geometric description defined. It should be noted none of the methods investigated truly change the systems model, yet CSV and XMI do not allow for the geometric model to be easily and instantly updated as new systems data becomes available. Conversely, the internal integration in SysML requires an ESP Model Library to be developed within the model and a system linkage to be created through the straight line association and parametric binding between the System Model and the Geometric Model shown in Figure 12 and Figure 13.

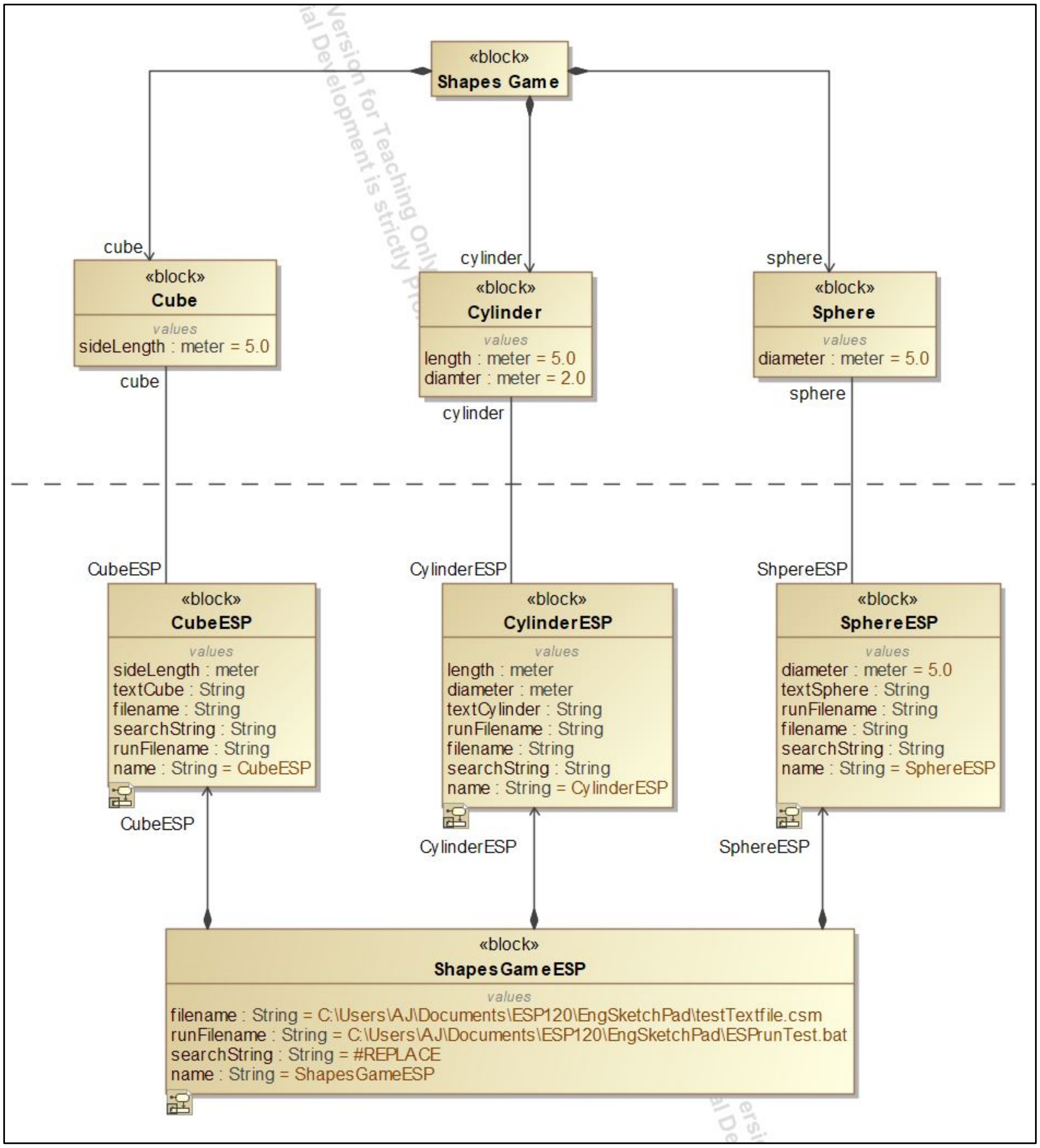
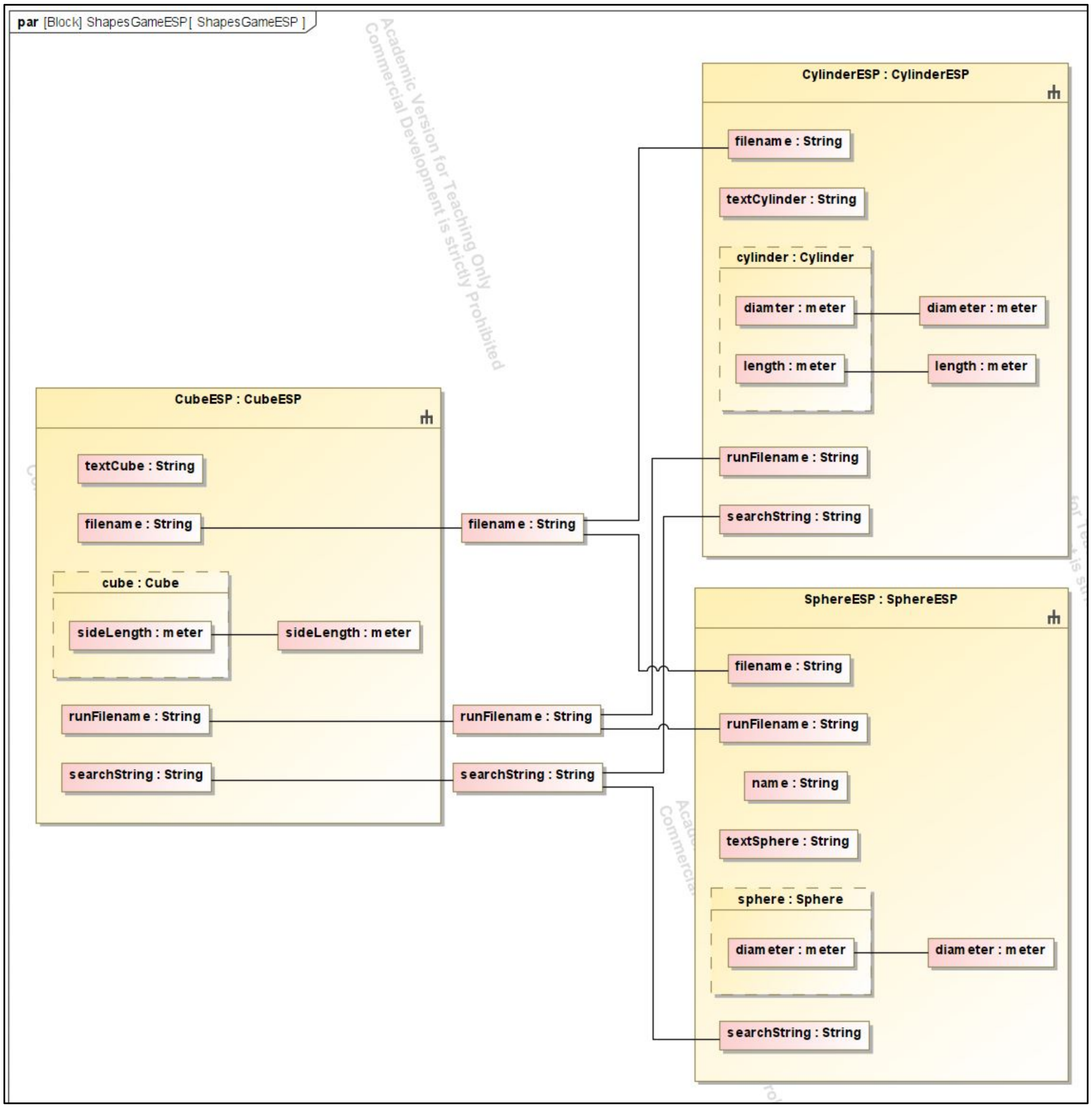


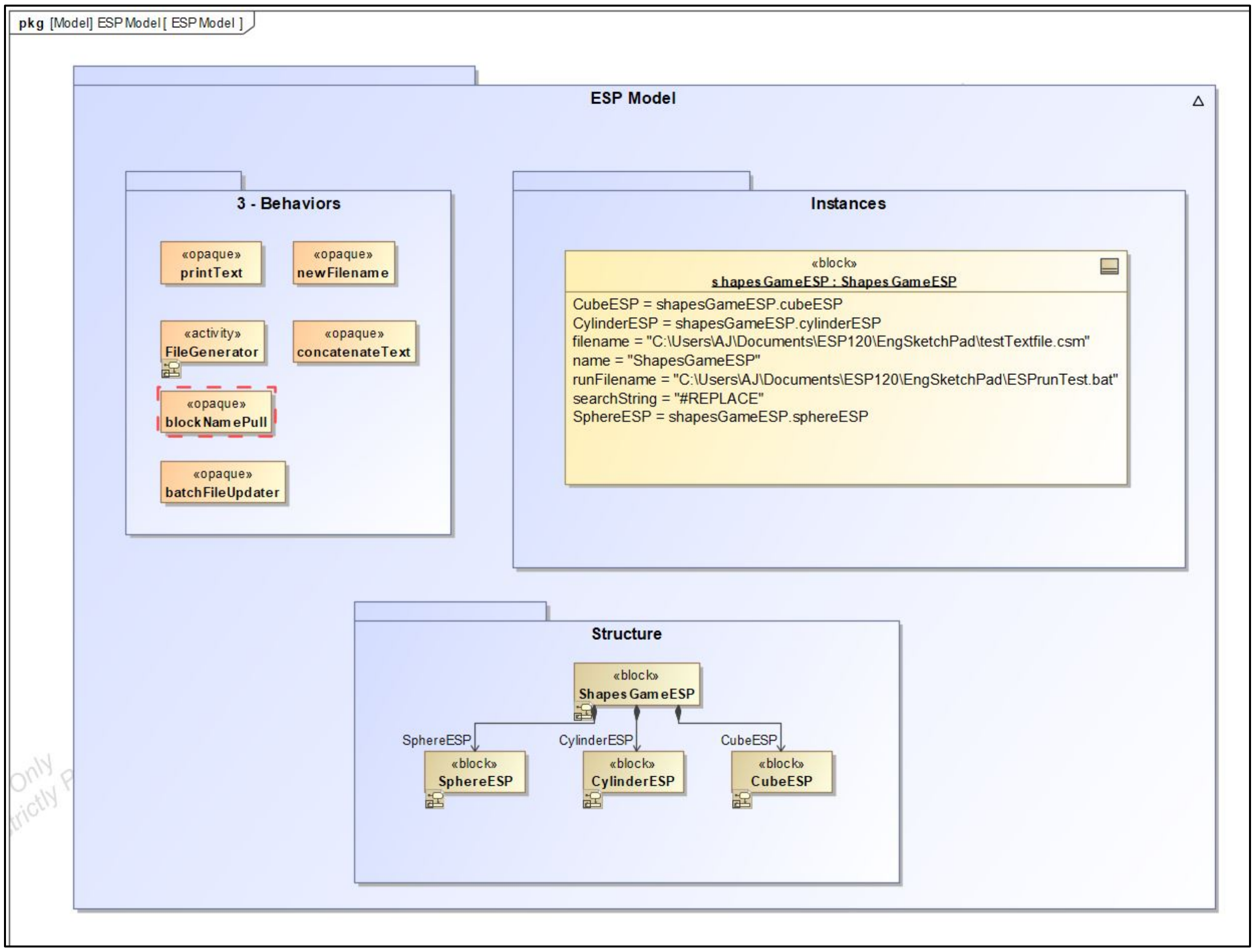
Figure 12: Straight Line Association Between System and ESP Models



Academic Version for Teaching Only
Commercial Development is strictly Prohibited

Figure 13: Parametric Diagram Example

Maintaining system design is critical to enabling this research, because requiring large changes to already existing systems will inherently lead to a lack of adoption of the technique and increased complexity in already complex systems which can cause difficult to overcome challenges. With that mentality at the forefront of this research, the method defined does not change the systems model but merely accesses the data which is already available, and in some cases may require simple value property additions or changes but does not impact the integrity of the systems model. This is achieved through the utilization of a separate package which contains a model library of the ESP related functionality required to geometrically define a system. This Package can be seen in Figure 14, which shows the containment of Behaviors, Instances, and Structure for the ESP Model Library.



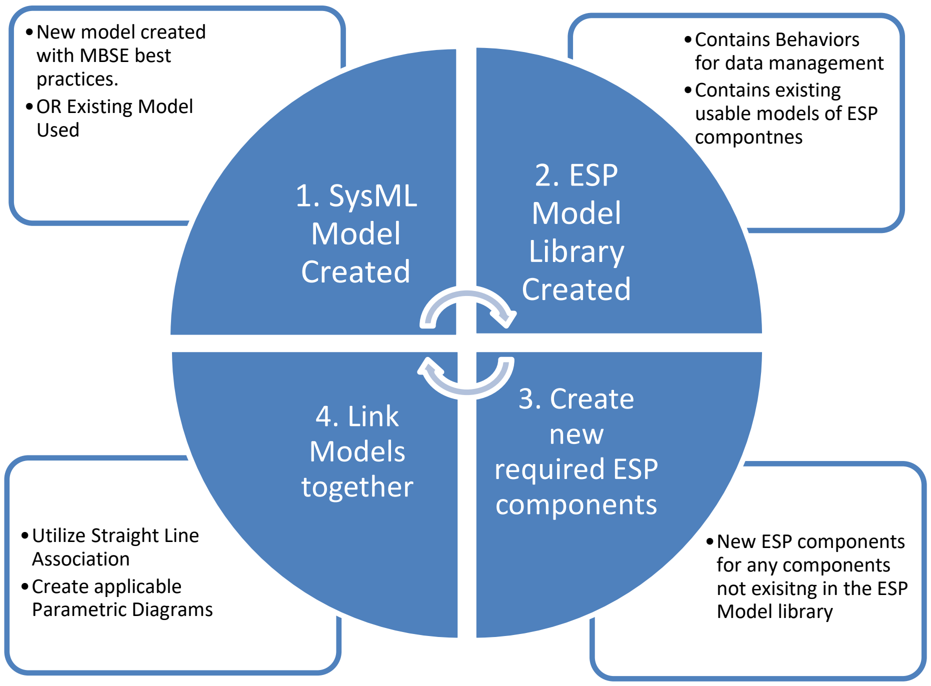
«block»
shapesGameESP : ShapesGameESP

CubeESP = shapesGameESP.cubeESP
CylinderESP = shapesGameESP.cylinderESP
filename = "C:\Users\AJ\Documents\ESP120\EngSketchPad\testTextfile.csm"
name = "ShapesGameESP"
runFilename = "C:\Users\AJ\Documents\ESP120\EngSketchPad\ESPrunTest.bat"
searchString = "#REPLACE"
SphereESP = shapesGameESP.sphereESP

Figure 14: Package Diagram of ESP Model Library

3.2.3 Research Question 3

The third research question identifies the pattern that is needed to integrate the SysML tool with the Geometric tool. It also strives to address the need for a decoupling of the System Model and the Geometric Model so that the original model or the new model being developed maintains integrity with currently defined systems modeling practices, namely architectural, physical, and functional decompositions. This pattern consists of four major steps which all are performed within the Systems Modeling Tool, Cameo Systems Modeler. Illustrated in Figure 15, the first step is to create a system model of the system desired or to utilize an existing model of a system to be explored. The second step is to create or import, if already existing, an ESP package which contains opaque behaviors and activities which define the data management process within the systems modeling tool for the geometric data. The third step is to create an ESP model library of the system, which contains blocks which described the geometric definition of system components, parametric diagrams which bind value properties within the ESP model library and ESP model itself to required value properties in the system model, and the activity diagrams which interface the ESP model library with the ESP package imported in the previous step. The 4th and final step in creating this pattern is to link the models together through a straight-line association.



• New model created with MBSE best practices.
• OR Existing Model Used
• Contains Behaviors for data management
• Contains existing usable models of ESP compontnes
1. SysML Model Created
2. ESP Model Library Created
3. Create new required ESP components
4. Link Models together
• Utilize Straight Line Association
• Create applicable Parametric Diagrams
• New ESP components for any components not exisitng in the ESP Model library

Figure 15: Basic Pattern for SysML to ESP Linking

This straight-line association ensures that both the ESP model and the system model have knowledge of each other's existence but are only referenced to each other and do not have the ability to change the definition of one another unless explicitly defined in a parametric diagram. Another note, for this pattern to properly work, the Simulation Profile provided within Cameo Systems Modeler was added as a Project Usage to the model for access to the already developed functionality. Specifically, Text File Read/Write Opaque Behaviors and Command Line Input behaviors which enable the data to be passed from the System Model, through the ESP model, to a format on which ESP can operate.

3.2.4 Research Question 4

The 4th and final research question is to demonstrate the integration between the system tool and the geometric modeling tool through a simple example. This simple example is defined as Shapes Game, where three simple shapes, a cube, a sphere, and a cylinder, are defined in a system model and then linked to an ESP model using the pattern described in Research Question 3. A detailed discussion on how the pattern was implemented through Shapes Game can be seen in Chapter Four of this thesis, but the basic steps identified through the exercising of this simple model can be seen in the following list:

1. Start a new SysML model
2. Create Basic "Shapes Game" model
3. Create Similar ESP model of "Shapes Game"
4. Apply necessary associations
5. Create required Opaque Behaviors
6. Create required activity diagrams for ESP blocks
7. Create required parametric diagrams for System to ESP data passing
8. Simulate Model
9. "Add Value" for reference properties
10. Store Instance of the ESP model
11. Simulate instance of ESP model to create required text files
12. Run .BAT file (internal or external) to visualize the model components

IV. Analysis and Results

4.1 Analysis and Results Overview

This research implements an applied approach (Assef, 2021). This means that rather than performing large quantities of calculations, the results are focused on a more application-based approach. To test whether the process is legitimate, it must be applied in a SysML environment, specifically Cameo Systems Modeler. This chapter focuses on that application in two ways. First, a simple ground-up approach was completed to ensure all steps in the process were validated and all functions performed as expected. This approach can be seen through the Shapes Game Example. Second, the process was applied to a preexisting systems model, namely a Rocket Model created in one of AFIT's Advanced Systems Modeling Courses. The two examples strive to show the process working in a manner which is applicable in real-world situations, and aids to identify failure points and areas for future research.

4.2 Shapes Game Example

To implement the process and methods described in Chapter 3, the first approach was to create a simple example to ensure the desired effects of each SysML modeling technique. The first step was to create a basic Shapes Game model consisting of three components, a Sphere, a Cylinder, and a Cube. These components are related to the Shapes Game Block through a Directed Composition, much like an aircraft is composed of its individual components. Figure 16 shows this basic model.

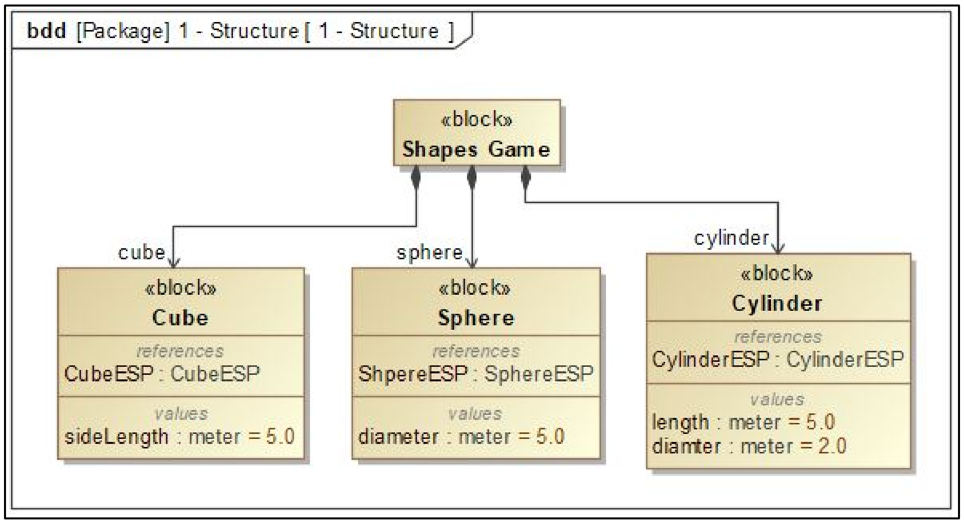


Figure 16: BDD of Shapes Game "Systems" Model

The next step is to create a similar model to Shapes Game, but within the ESP Model Library. This model, shown in Figure 17, will be the touch point between Cameo and ESP to enable the system and geometric information to be passed to a CSM file and then executed through OpenCSM.

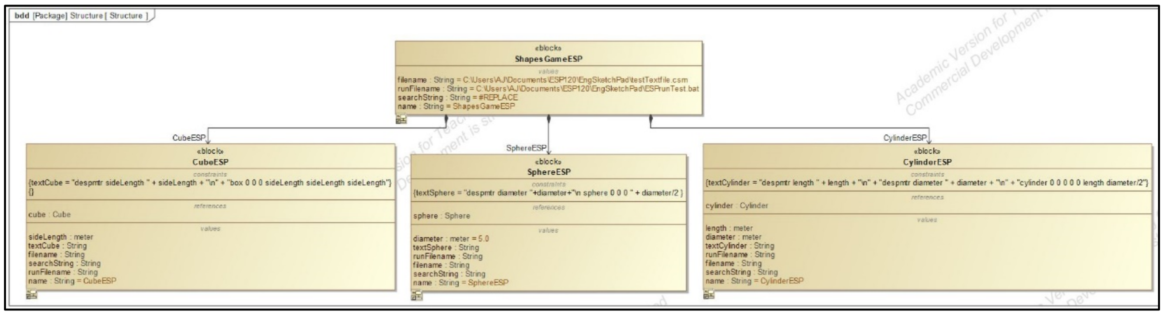
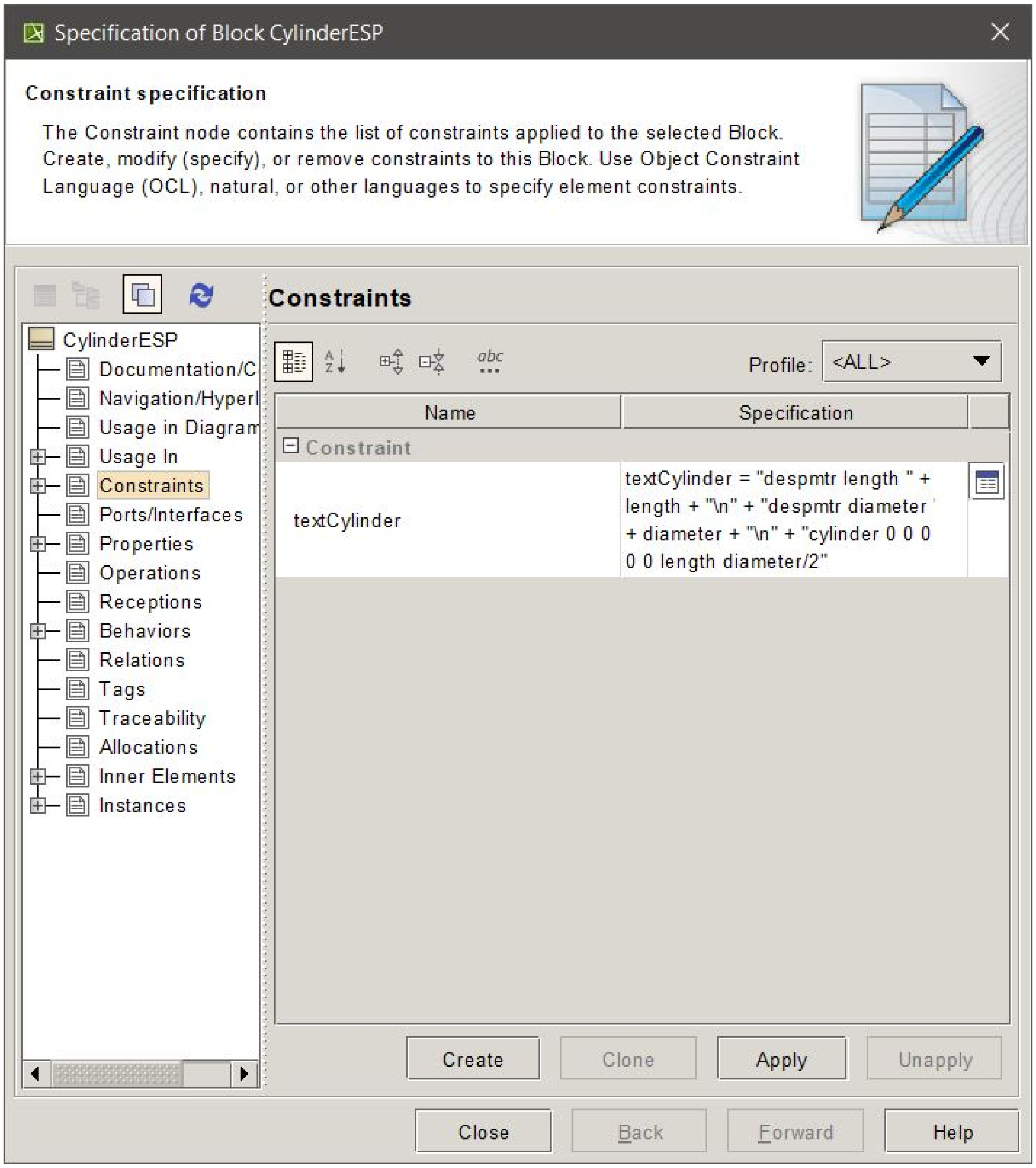

bdd [Package] Structure [Structure]
«block»
ShapesGameESP
values
filename : String = C:\Users\AJ\Documents\ESP120\EngSketchPad\testTextfile.csm
runFilename : String = C:\Users\AJ\Documents\ESP120\EngSketchPad\ESPrunTest.bat
searchString : String = #REPLACE
name : String = ShapesGameESP
CubeESP
«block»
CubeESP
constraints
{textCube = "despmtr sideLength " + sideLength + "\n" + "box 0 0 0 sideLength sideLength sideLength"}
{}
references
cube : Cube
values
sideLength : meter
textCube : String
filename : String
searchString : String
runFilename : String
name : String = CubeESP
SphereESP
«block»
SphereESP
constraints
{textSphere = "despmtr diameter "+diameter+"\n sphere 0 0 0 " + diameter/2 }
references
sphere : Sphere
values
diameter : meter = 5.0
textSphere : String
runFilename : String
filename : String
searchString : String
name : String = SphereESP
CylinderESP
«block»
CylinderESP
constraints
{textCylinder = "despmtr length " + length + "\n" + "despmtr diameter " + diameter + "\n" + "cylinder 0 0 0 0 0 length diameter/2"}
references
cylinder : Cylinder
values
length : meter
diameter : meter
textCylinder : String
runFilename : String
filename : String
searchString : String
name : String = CylinderESP
Academic Version for Teaching Only
Commercial Development is strictly prohibited

Figure 17: Shapes Game ESP BDD

Each Component in the ESP Model has unique values which must exist for the model to properly execute. These required value properties are SearchString, Filename, RunFilename, Name, and textComponent. SearchString is a Value Property which defines what string combination should be found in the default text file and then replaced with the ESP text which describes the ESP Component. Filename is the name of the CSM file which is being created for the ESP Component in question, whereas RunFilename is the name of the Windows Batch File which enables ESP to be executed automatically. Name is the Value Property which describes the name of the individual component so that name-based properties and filenames can be created. Finally, textComponent is a value property which is described through a Constraint Expression in the ESP block. This expression describes the CSM syntax which creates the component in question. An example of this Constraint Expression is shown in Figure 18.


Specification of Block CylinderESP

Constraint specification

The Constraint node contains the list of constraints applied to the selected Block. Create, modify (specify), or remove constraints to this Block. Use Object Constraint Language (OCL), natural, or other languages to specify element constraints.

Constraints

Name	Specification
Constraint	
textCylinder	textCylinder = "despmtr length " + length + "\n" + "despmtr diameter " + diameter + "\n" + "cylinder 0 0 0 0 0 length diameter/2"

Figure 18: ESP Text as Constraint Expression

This method utilizes the ability of Constraint Expressions to access the Value Properties of the owning Block and automatically modifies the script for the final ESP text when a change is made. Currently, these Constraint Expressions are generated with “English” as the language for the Body. Once each side of the model is created, the associations and parametric diagrams must be made to properly declare how the information is owned, managed, and handled. As discussed in Chapter 3, this is achieved through a Straight-Line Association and Parametric Diagrams. The merging of these two models can be seen in Figure 19.

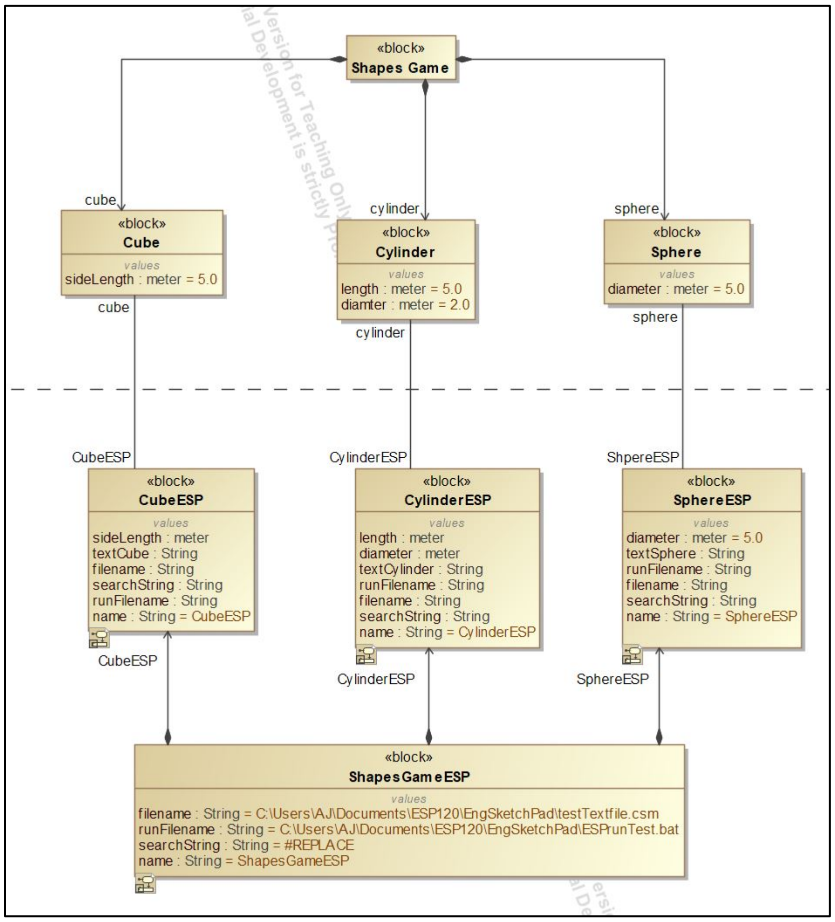


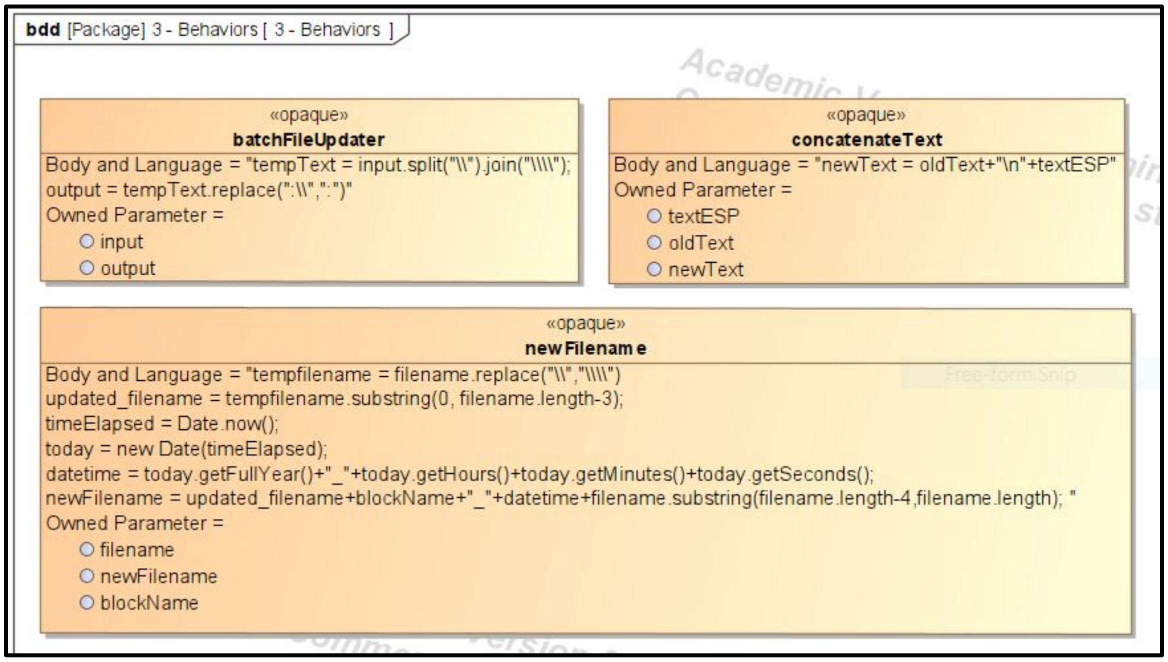
Figure 19: BDD of Shapes Game and ESP Integrated Model

After the model associations are properly defined, the opaque expressions and activities must be made to handle the information in the model. These expressions aid in handling the data in the ESP Blocks and uses them to update the pre-existing files with the correct information. The pre-made Opaque Behaviors which are used from the Simulation Profile

Package are TextFileReplace and CommandLineInput. The custom Opaque Behaviors used are described in Table 1 and are detailed in Figure 20.

Table 3: Opaque Behaviors for ESP Model Generation

Opaque Behavior Name	Description
batchFileUpdater	An opaque behavior designed to handle the updates for text in the Windows Batch File
concatenateText	An opaque behavior which combines the text for each component into a single CSM file for the model to be run in a single CSM
newFilename	An opaque behavior which updates and creates a unique CSM and Windows Batch filename for each component



```
bdd [Package] 3 - Behaviors [ 3 - Behaviors ]

«opaque»
batchFileUpdater
Body and Language = "tempText = input.split("\\").join("\\\\");
output = tempText.replace(":\\",":")"
Owned Parameter =
    input
    output

«opaque»
concatenateText
Body and Language = "newText = oldText+"\n"+textESP"
Owned Parameter =
    textESP
    oldText
    newText

«opaque»
newFilename
Body and Language = "tempfilename = filename.replace("\\","\\\\")
updated_filename = tempfilename.substring(0, filename.length-3);
timeElapsed = Date.now();
today = new Date(timeElapsed);
datetime = today.getFullYear()+"_"+today.getHours()+today.getMinutes()+today.getSeconds();
newFilename = updated_filename+blockName+"_"+datetime+filename.substring(filename.length-4,filename.length); "
Owned Parameter =
    filename
    newFilename
    blockName
```

Figure 20: Opaque Expressions for ESP Data Management

The activities which are needed to execute these Opaque Expressions and Behaviors are included in the ESP Model Library. The main activity at the core of these models is the TextFileGenerator, which takes the inputs from the ESP Model Components and then creates a text file which contains the ESP required text to describe the geometry of the component. This Activity can be seen in Figure 21, which shows the inputs of searchString, textESP, filename, runfileLocation, and blockName.

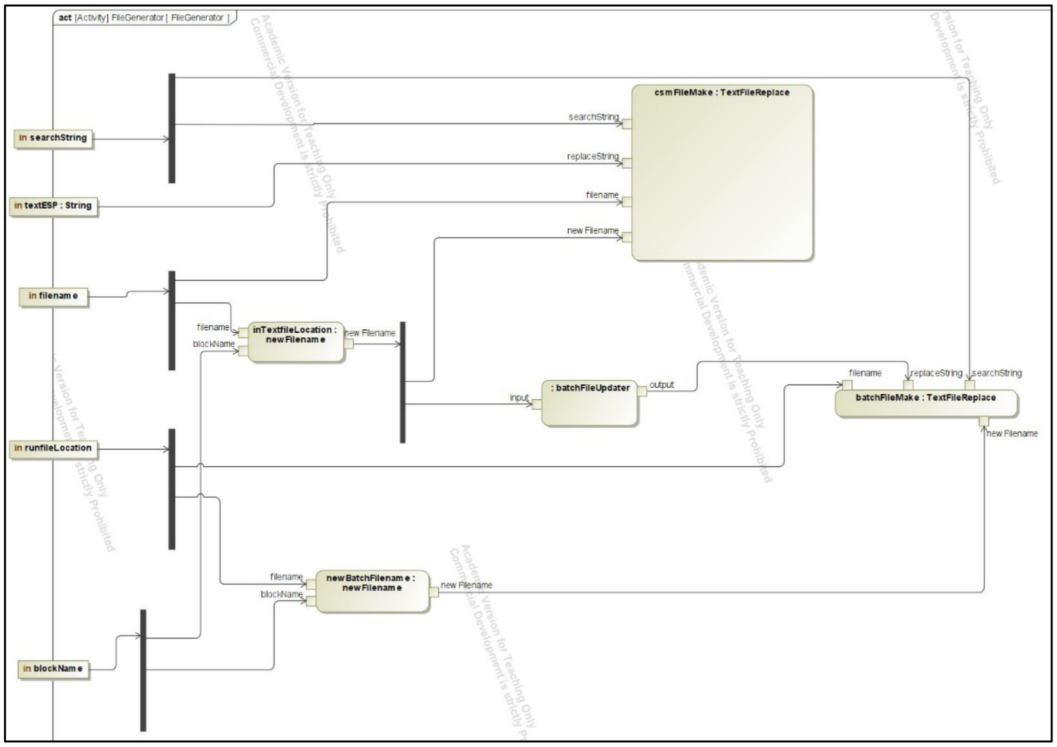


Figure 21: FileGenerator Activity

This Activity is used as the final activity for each of the Owned Behaviors of the ESP Components. An example of this activity is shown in Figure 22, where the readSelf Action is used to access the CubeESP object and create an Object Flow representing CubeESP to each of the readStructuralFeature actions. The readStructuralFeature action allows for the Value Properties of the individual ESP components to be accessed and then passed to the FileGenerator Activity.

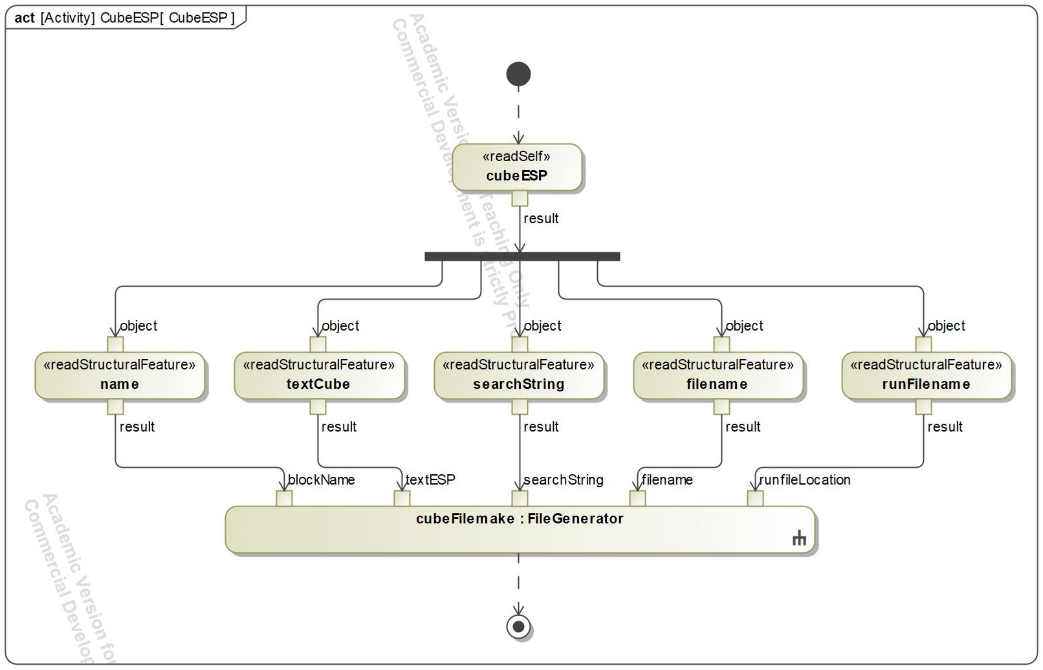
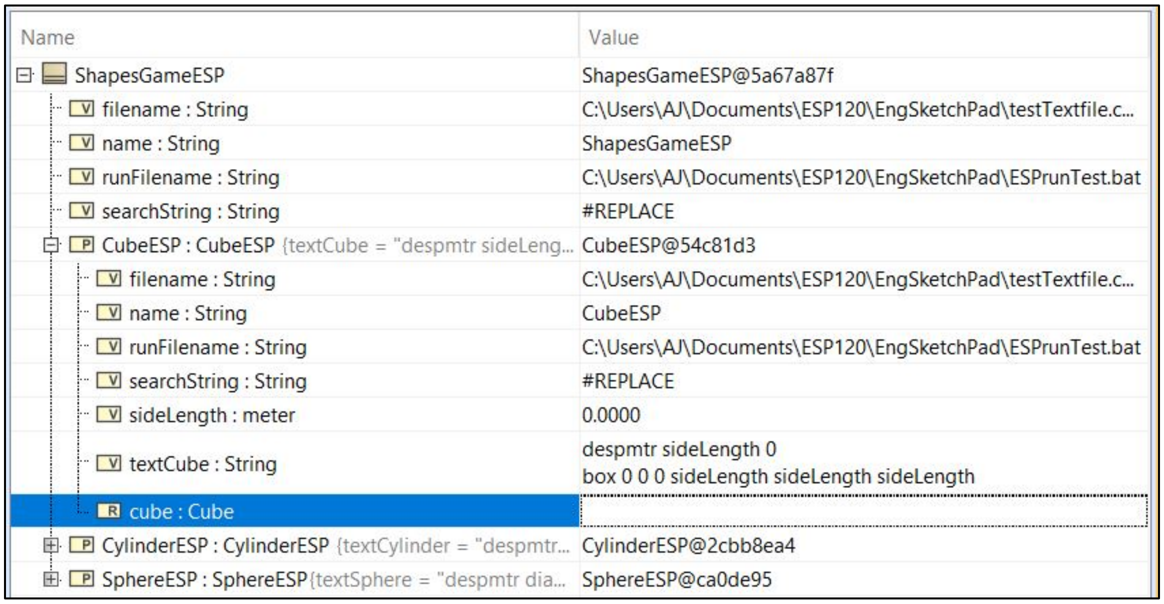


Figure 22: cubeESP Owned Behavior Activity Diagram

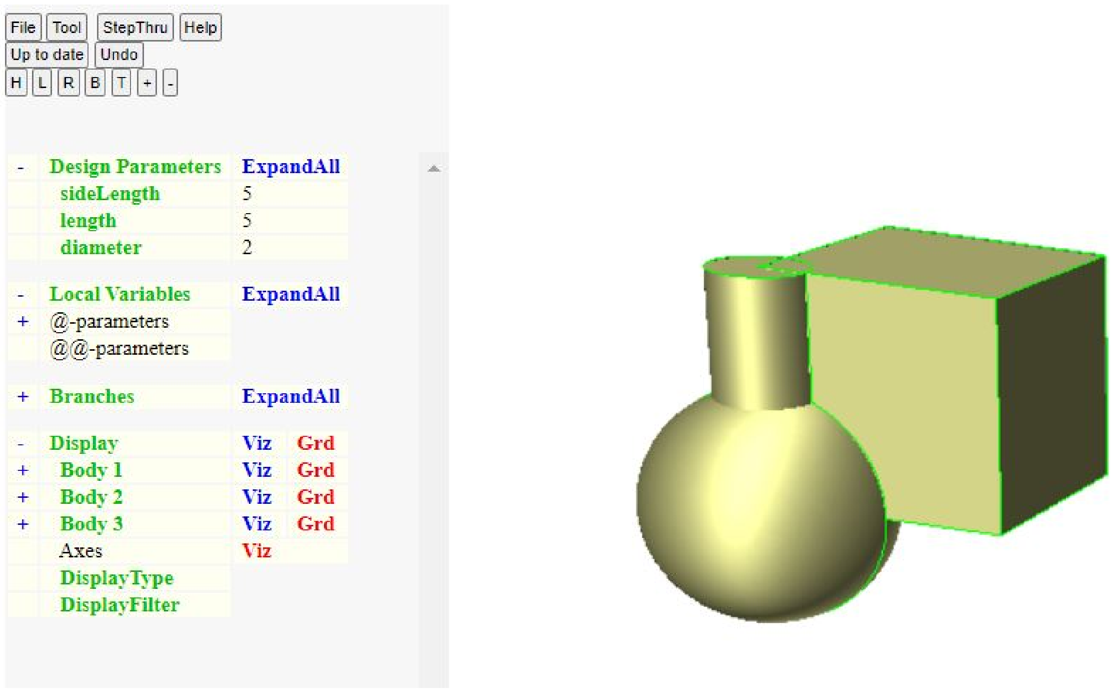
At this point in the process, the model has been fully developed and the relationship between the System Model and the ESP Model has been defined. The next step is to Simulate the Shapes Game ESP Block, which acts similar to an Analysis Context block, and save that simulation as an instance. The first step in the Simulation is to Add Value to each of the Reference Properties in the ESP Components, shown in Figure 23.



Name	Value
ShapesGameESP	ShapesGameESP@5a67a87f
filename : String	C:\Users\AJ\Documents\ESP120\EngSketchPad\testTextfile.c...
name : String	ShapesGameESP
runFilename : String	C:\Users\AJ\Documents\ESP120\EngSketchPad\ESPrunTest.bat
searchString : String	#REPLACE
CubeESP : CubeESP {textCube = "despmtr sideLeng...	CubeESP@54c81d3
filename : String	C:\Users\AJ\Documents\ESP120\EngSketchPad\testTextfile.c...
name : String	CubeESP
runFilename : String	C:\Users\AJ\Documents\ESP120\EngSketchPad\ESPrunTest.bat
searchString : String	#REPLACE
sideLength : meter	0.0000
textCube : String	despmtr sideLength 0 box 0 0 0 sideLength sideLength sideLength
cube : Cube	
CylinderESP : CylinderESP {textCylinder = "despmtr...	CylinderESP@2cbb8ea4
SphereESP : SphereESP{textSphere = "despmtr dia...	SphereESP@ca0de95

Figure 23: Simulation Output of ShapesGameESP Context Block

This is done by simply Right Clicking and then selecting the "Add Value" menu option. Then an instance of Shapes Game ESP can be saved, and that instance can be simulated to create the desired text files. An example of the final text files can be seen in Appendix A. The final output of this process is the ESP Model, which is a geometric representation of the Shapes Game System. This representation is very simple and lacks some information which is useful in a larger system model such as component positioning (i.e., every component is built and stays at the Cartesian (0,0,0) position). Nonetheless, Figure 24 shows that the Shapes Game System can be seen in Geometric Form through the proper management of Systems Data within SysML, the creation of a few custom Opaque Expressions, and proper use of Association types.



```
-  Design Parameters   ExpandAll
     sideLength        5
     length            5
     diameter          2

-  Local Variables     ExpandAll
+  @-parameters
   @@-parameters

+  Branches            ExpandAll

-  Display             Viz   Grd
+    Body 1            Viz   Grd
+    Body 2            Viz   Grd
+    Body 3            Viz   Grd
     Axes              Viz
     DisplayType
     DisplayFilter
```

Figure 24: ESP Output of ShapesGameESP

4.3 Utilization of an Existing Model

To further explore this process and to ensure the methods described in Chapter 3 and Chapter 4.1 hold true in an existing model, a Model which was created by a separate user as part of AFIT's SENG 660 Advanced Systems Modeling Class was collected and used as a starting point for this next section. The first step was to select a Model which contained enough Complexity to stress the process, without making the manual processes more important than some of the exploratory findings regarding the System to Geometric Interface. The model which was selected was a basic rocket model created by Captain Patrick Assef (Assef, 2021). This model contains many different and irrelevant components (as far as Geometric Description is concerned), so it is a good test for the usefulness of the processes/pattern previously described. The overall process was exactly the same as the process described for Shapes Game with two major differences. First, the necessity for positioning of the individual components was now crucial, which led to having multiple System blocks associated to the same ESP component. The primary purpose of this association, shown in Figure 25, is to provide a way for each component to have knowledge of the geometric properties of other system blocks. This also reduces the need for duplication of Value Properties across multiple components (i.e., storing information in one block at the System Level as opposed to multiple at the ESP level). In practice, this allows for components which are either geometrically constrained or dependent on other components to be generated without the need for repetitive value creation.

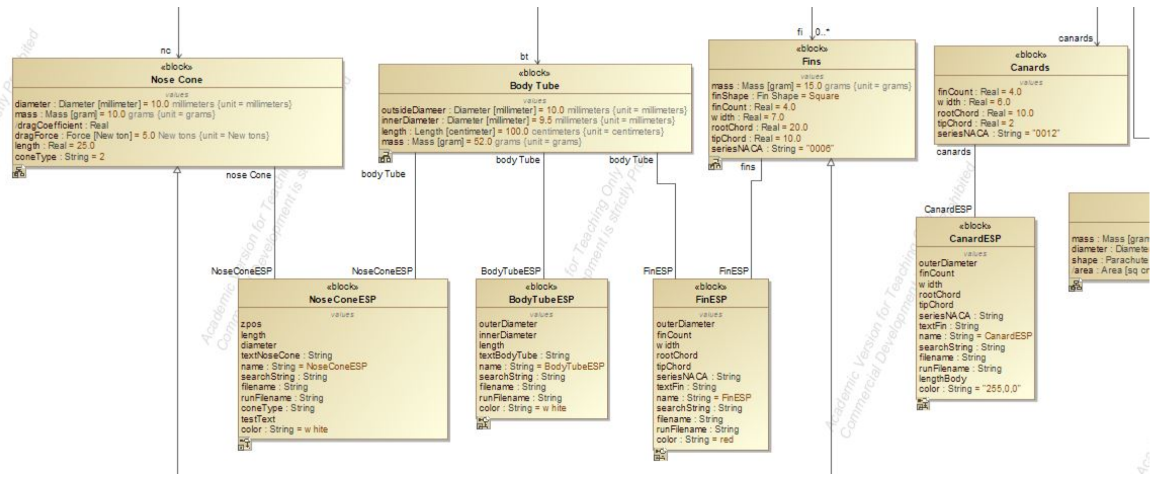
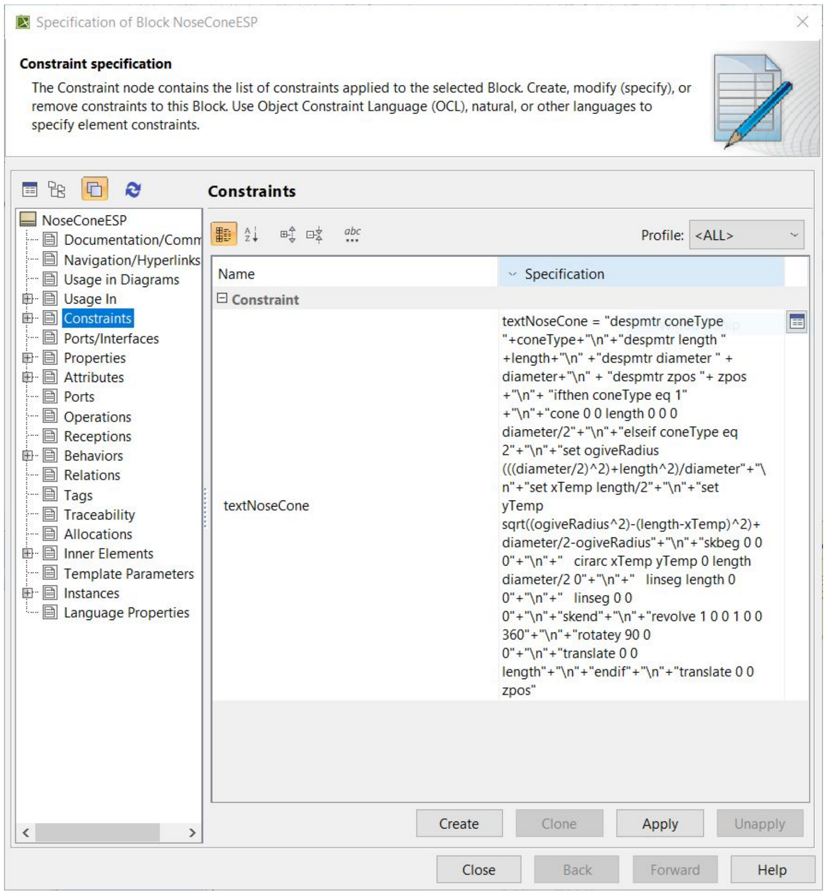


Figure 25: Rocket Model Physical Decomposition Showing Multiple Association Paths

The second difference between the Rocket Model and the Shapes Game model is that the Behaviors and the basic ESP components were imported through the Model Library Created in Shapes Game. This eliminated the need for a large portion of the work from Shapes Game to be redone within the Rocket Model. This method still requires some manual connecting of the System and Geometric Components as well as the pre-made text files for the ESP and Windows Batch Files to be made. Another note, as stated before, the Add Value process does get tedious as more components and more complex relationships are developed. While this can become tiresome, it is believed that this process enables more future work than it hurts, especially for systems early in their lifecycle. Along with the major differences between the two implementations of the process from Chapter 4, there were many components which contained significantly more detail than the Shapes Game Model. For example, the Constraint Expressions used for each of the ESP components is much more complex in the Rocket Model than it is in the Shapes Game Model, shown in Figure 26. Compared to the Constraint Expression in Figure 18, it can be seen the level of ESP usage is much higher to enable the more complex shapes found in the Rocket Model.



Specification of Block NoseConeESP

Constraint specification

The Constraint node contains the list of constraints applied to the selected Block. Create, modify (specify), or remove constraints to this Block. Use Object Constraint Language (OCL), natural, or other languages to specify element constraints.

Constraints

Profile: <ALL>

Name	Specification
Constraint	
textNoseCone	textNoseCone = "despmtr coneType "+coneType+"\n"+"despmtr length " +length+"\n" +"despmtr diameter " + diameter+"\n" + "despmtr zpos "+ zpos +"\n"+ "ifthen coneType eq 1" +"\n"+"cone 0 0 length 0 0 0 diameter/2"+"\n"+"elseif coneType eq 2"+"\n"+"set ogiveRadius (((diameter/2)^2)+length^2)/diameter"+"\n"+"set xTemp length/2"+"\n"+"set yTemp sqrt((ogiveRadius^2)-(length-xTemp)^2)+ diameter/2-ogiveRadius"+"\n"+"skbeg 0 0 0"+"\n"+" cirarc xTemp yTemp 0 length diameter/2 0"+"\n"+" linseg length 0 0"+"\n"+" linseg 0 0 0"+"\n"+"skend"+"\n"+"revolve 1 0 0 1 0 0 360"+"\n"+"rotatey 90 0 0"+"\n"+"translate 0 0 length"+"\n"+"endif"+"\n"+"translate 0 0 zpos"

Figure 26: Rocket Model Nose Cone ESP Constraint Expression

Along with the individual complexities of the Rocket Model Components, it was critical to enable the entire Geometric Model to be displayed at once. To enable this, a separate activity owned by the Rocket Model ESP block must be created. This activity can be seen in Figure 27.

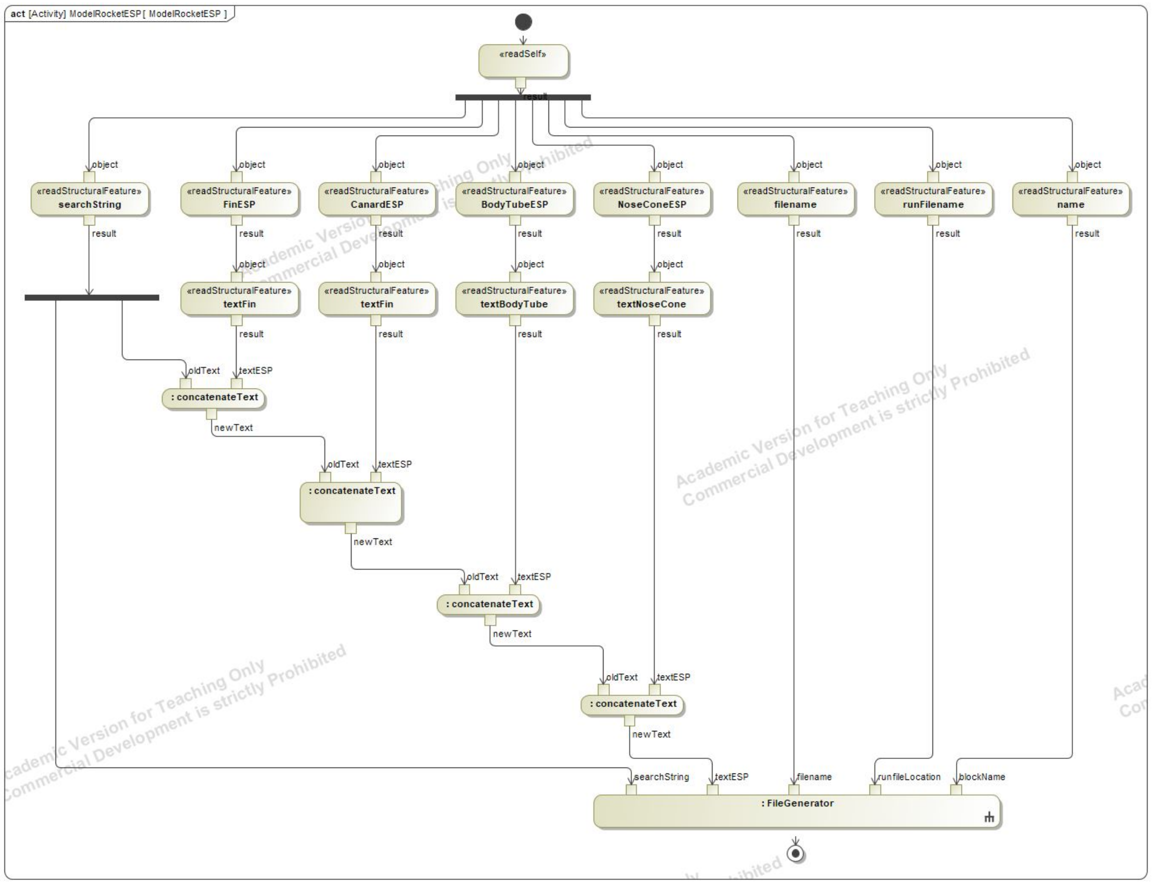

act [Activity] ModelRocketESP[ModelRocketESP]
«readSelf»
result
object
«readStructuralFeature» searchString
«readStructuralFeature» FinESP
«readStructuralFeature» CanardESP
«readStructuralFeature» BodyTubeESP
«readStructuralFeature» NoseConeESP
«readStructuralFeature» filename
«readStructuralFeature» runFilename
«readStructuralFeature» name
«readStructuralFeature» textFin
«readStructuralFeature» textFin
«readStructuralFeature» textBodyTube
«readStructuralFeature» textNoseCone
oldText
textESP
: concatenateText
newText
searchString
filename
runfileLocation
blockName
: FileGenerator

Figure 27: RocketModelESP Owned Activity

This activity is intended to extract the ESP text created by each individual component and pass it to a final overall ESP file which can build all components together. This is accomplished through the use of the concatenateText Opaque Behavior, which builds the final ESP file iteratively for each component. There is concern that this would become a manually intensive activity to build in more complex systems, and potential solutions to this issue are presented in Chapter 5. The final product of the Rocket Model can be seen in Figure 28.

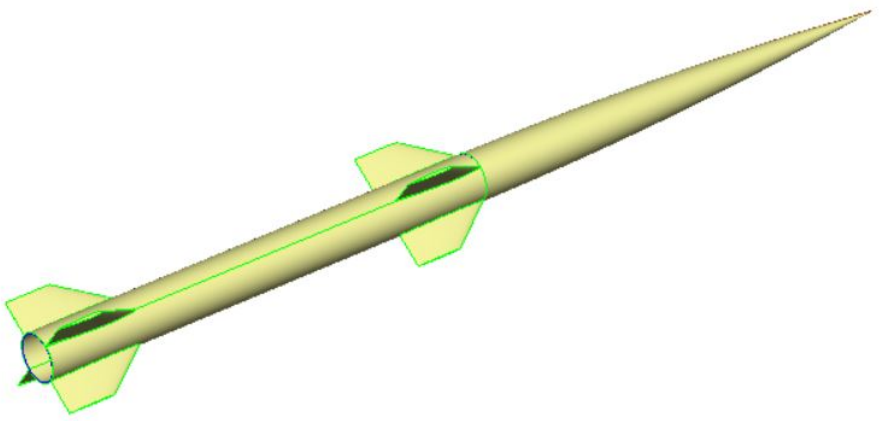
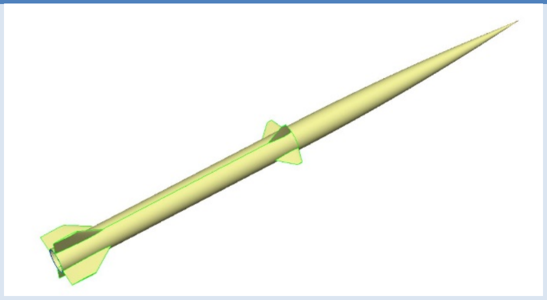
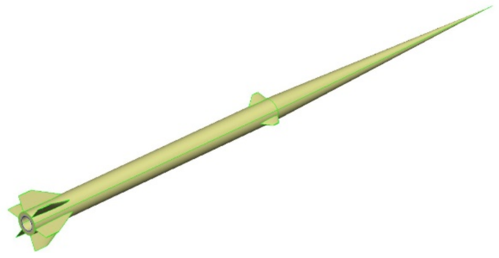
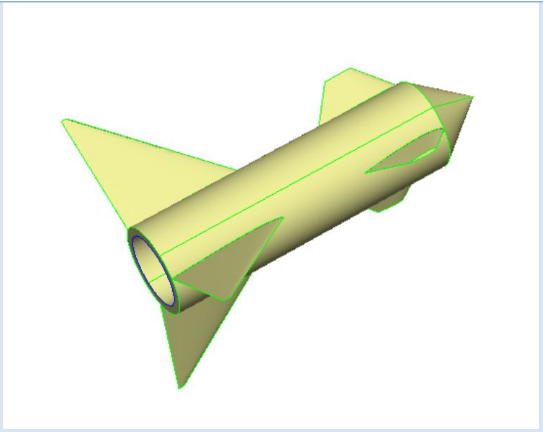
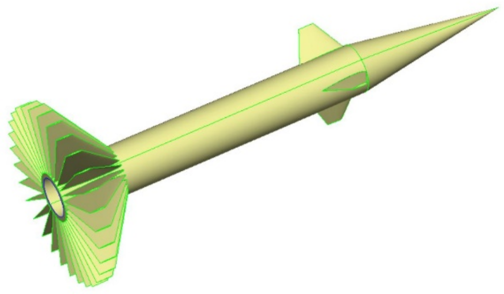


Figure 28: ESP Output of Rocket Model

This image is a basic output which can be found from ESP for the Rocket Model. To explore what is potentially possible, some variations from the rocket were created. This was done through the creation of multiple RocketModelESP instances, stored within the ESP Model Library. These variations are shown in Table 4.

Table 4: Rocket Model Variations

Rocket Variations	Differences
	Reduced to three Canards and a longer overall Body Tube and Nose Cone

	Increased to six fins and different values for Canard and Fin description
	Decreased nose length, increased fin size and count, decreased Body Tube length and inner diameter
	Increased Nose Cone length, Fin size and count, and Body Tube length

This table shows two basic shapes with minimal changes and then two extreme shapes with major changes. This is intended to show if the capability of SysML to perform Trade Studies and Monte Carlo analysis is utilized, the ESP Model should be able to handle the variations with relative ease.

V. Conclusions and Recommendations

5.1 Introduction of Research

This thesis strives to address the stovepiped gap related to Geometric Generation and Systems Modeling through MBSE tools. This gap is created through a lack of current patterns and processes, as well as lack of proper tools knowledge, which causes discipline engineers to create models in their own environments which are not fully based on a single truth model. This research addressed the existence of this gap, the need for a bridge between the two toolsets, and provided an approach to create that bridge which also opens the door to many avenues of potential future research and development regarding increasing the capabilities and effectiveness of our MBSE and Systems Engineering processes. Each of the four Research Questions addressed throughout this thesis contribute to the development of a process which 1) Is created in a decoupled nature to maintain the integrity of the System Model, 2) Provides a way to interface between a Geometric Model and a Systems Model without requiring changes to the overall System Model, and 3) Is created in a way such that future research can build upon the capabilities of BOTH SysML and ESP.

5.2 Summary of Research Gap, Research Questions and Answers

The research gap addressed here is one of model integration between the disciplines of Systems Engineering and Geometric Modeling. Specifically, the importance of removing the stovepiped gap generated by different "truth" models for a system and how removing that gap addresses the drive towards a Digital Twin environment. The elimination of this gap leads to more accurate models, models which are directly derived from requirements and system descriptions, and potentially ties in the ability for future integration of high-fidelity analysis tools. Performing the task of removing this gap early in a system's lifecycle is paramount to the system being able to properly handle the current paradigm shift towards a Digital Engineering world. The answers to each of the 4 research questions has been detailed in Chapters 3 and 4. A summary can be found in the following sections.

5.2.1 Answer to Research Question 1

The first research question addresses the need to identify difficulties in integrating SysML with analysis-centric geometry. The major areas of concern relative to this integration are the need for the system data to be available to the geometric model and model definition. This was addressed through exploration of multiple different data management pathways (CSV, XMI, and Internal), and identified that the most logical way to handle the data pathways was within the current capabilities of SysML.

5.2.2 Answer to Research Question 2

The second research question is related to the need for the integration to be seamless between the geometric model and the system model. This question assumes that either A) the model being developed is being owned and created through SysML or B) the model in question already exists within a SysML framework. This need for minimal change and coupling of the two models is paramount to enabling this process for a wide spectrum of use cases. To address this need, the exploration of association types and use of parametric diagrams was performed, and it was concluded that the use of a Straight-

Line Association enables access to System and Geometric Model information but reduces the coupling of the two systems.

5.2.3 Answer to Research Question 3

This research question is the core of this thesis. The development of a method for integration, rather than a tool which links the systems, is paramount in understanding how the System Model and ESP Model behave when linked. To enable this, a basic pattern was developed which focuses on how the data in the system should flow and how this can be done to reduce the overall change to the System Model. The final pattern used is:

1. SysML Model Created
2. ESP Model Library Created
3. Create new required ESP components
4. Link Models together

5.2.4 Answer to Research Question 4

The fourth and final research question is one of application. To ensure the pattern and processes outlined in Chapter 3 hold true, they must be implemented in a System Model. The two models used were Shapes Game and the Assef Rocket Model, and the details of these implementations can be seen in Chapter 4.

5.3 Study Limitations

There were a few limitations applied to this research. The primary limitation is one of tool choice. The tools selected for this research were Cameo Systems Modeler (now Catia Magic Systems of Systems Architect) and Engineering Sketch Pad. These tools were selected for various reasons, primary of which was ease of access. Cameo is a widely taught and widely used tool in the Department of Defense in regard to Systems Engineering, and it bodes well for adoption of this research if this tool is utilized. ESP is an open-source tool, which can be downloaded directly from a Massachusetts Institute of Technology website. The other limitations for this research were ones of availability and knowledge. SysML 2.0, while on the horizon, is not yet available, so it was assumed that this research enables the capabilities found in SysML 1.6 only. Also, for any code-based efforts, Python was the language of choice based on user knowledge and Python capabilities. The final limitation placed on this research was distribution limitations. All data generated and used here was generated under the assumption of UNCLASSIFIED publicly available data. This enables the research to be utilized as widely as possible as a trade for more specific DoD-related implementation.

5.4 Recommendations for Future Research

As this research is assumed to be the first implementation of Geometric Design directly merged with SysML, there are many areas which can be addressed in the near future. The main areas which should be considered are listed here:

1) Improvement of Data Management and Routines within the current pattern
2) Data Management and Information Feedback to SysML
3) Improvement of ESP interface for advanced functions
4) Analysis of Geometric Bodies output through this process

5.4.1 Improvement of Data Management and Routines

This area of research should focus on a few main points within the current process and attempt to improve them to enable future research. First, a majority of the Opaque Expressions are extremely basic. This is partly the nature of the complexity of the problem they attempt to solve, but also is an artifact of ensuring the researcher was able to identify failure points within the model simulations. In theory, these Opaque Expressions (namely batchFileUpdater and newFilename) could be consolidated into a single expression which has the necessary inputs and outputs. Second, the current implementation of this process utilizes "English" as the language for the CSM text. This is due in major part to the shortcomings of the embedded compiling capabilities in SysML tools. For example, there are many deprecated or unusable functions native to Jython and JavaScript which struggle to run in the SysML environment. It is recommended that a better way be developed to properly store the ESP script text within the Constraint Expression to eliminate the need for constant newline ("\n") instances. Ideally, the basic ESP script without the actual Property Values written in could be generated separately and then copied into the Constraint Expression for evaluation. Currently, this process is tedious and can be cause for much headache in for more complex geometries. The last area of research in this section would be to investigate a better way to extract and handle the Value Properties of each ESP Block. Currently, the easiest and most effective way found was to make use of the readSelf and readStructuralFeature Actions. These actions, while extremely useful, become tedious to handle at best, as a new readStructuralFeature is needed for each value to be assessed. It would be ideal for a better method to be developed, specifically one which utilizes the native dot notation for SysML hierarchies. The built-in Action Language Helper could also solve the problem here but was bypassed due to it being a Cameo exclusive method.

5.4.2 Data Management and Information Feedback to SysML

Another area of future research is the development of a method to pass data generated by the ESP Model back to the Systems Model. The first step to this would be to extract and then pass back the values relative to a Component and the total system which are generated by ESP. These include, but are not limited to: Volume, Geometric CG, Surface Area, Maximum and Minimum Values, and the specific Value Properties (or Design Parameters in ESP) used to build the model. This data passing can be achieved through the use of either A) and script which reads through and extracts the data in an EGADS file for the model or B) through the CAPS or pyCAPS interfaces which are utilized to query the model for information and access the ESP built in APIs. After the desired information is extracted from ESP, a method to pass data back to the SysML model must be created. There are ways to pass data directly to SysML through CSV files, but they were not explored in this thesis to the extent to be sure of its validity. The next step would be to define the ESP model and Systems Model to properly handle the new data. Ideally, all the data generated in ESP is stored in the ESP Model and then only necessary information is passed to the Systems Model.

5.4.3 Improvement of ESP interface for advanced functions

One area of ESP which was not fully explained or explored were the more advanced functions such as Booleans (i.e., Subtract, Union, Join, Combine), Grown bodies (i.e., Rule, Extrude, Blend), and some of the more complex Declarations (i.e., attribute, name, outpmtr, dimension). These functions truly aid in increasing the capability of ESP to be a more than formidable tool in a modeler’s arsenal. The challenge here is that the more complex the design, the more each component requires information from the others. For example, to blend a complex fuselage to a conical nose cone, one could utilize the Blend or Rule functions. This requires certain properties of each body to be known, such as face numbers, edge numbers, and edge quantity to be known and a specific order to be picked so that surface normals are facing the correct way. For example, two wire bodies with differing edge counts cannot be ruled together, and ESP will error out on this attempt. Attempting to bridge this gap with significantly aid in the flexibility of the modeler to create more complex shapes. It can be postulated that these methods can be handled through specific activities within the ESP Model Library. For example, if two bodies are to be ruled together, an activity which imports them both and then queries them in the right order could be developed. This may take some interaction with the ESP Development Team (MIT, Syracuse, and AFRL/RQV) to properly merge this gap.

5.4.4 Analysis of Geometric Bodies

The final future research area would be to analyze the Geometric bodies being developed through this process. ESP comes packaged with several APIs (called Analysis Interface Modules (AIMs) in CAPS) which interface with commonly used analysis tools. Along with that, ESP is capable of exporting into many different grid formats which can be used in a variety of analysis efforts. An exploration should be done on how to automate which analysis should be done, how to properly handle the information needed, and similar to Section 5.4.2, how to pass that information back to the Systems Model.

This research could be the most beneficial in the future in enabling a truly Digital Twin/Digital Thread environment for physical systems being acquired by the DoD.

5.5 Summary or Significance of Research

This research is a major stepping-stone in realizing the DoD and USAF goal of a Digital Twin/Digital Thread environment. The desire to eliminate the stovepipes within which current design and engineering is done must be paired with an equal urge to improve upon the capabilities already in existence. SysML is a powerful tool in a Systems Engineer’s arsenal. Geometric design is a necessary step in providing a fully realized systems model and aiding in the decision-making process for design choices. The merging of these two disciplines opens the door for many possibilities in the world of Digital Engineering. If adopted, this method could prove to be a catalyst for future efforts in the areas of Digital Test and Evaluation, Digital Logistics Planning, and even enables more comprehensive models for Campaign Modeling. Only the future can tell how far this line of research may go, but it is the hope of this researcher that it is taken as far as it can be. To quote one of the final lines from one of my personal favorite television shows, Stargate SG-1, “Nothing Ventured, Nothing Gained.”

Bibliography

Assef, P. (2021). *SENG 660 Rocket Model.* Assef.

Buede, D. M. (2016). *The Engineering Design of Systems: Models and Methods.* Hoboken, NJ: Wiley.

Clough, J. (2019). Automated Wing Internal Structure Placement Guided by Finite Element Analysis. *AIAA Aviation 2019 Forum* (p. 13). AIAA.

Colombi, J., Miller, M. E., Schneider, M., McGrogan, J., Long, D. S., & Plaga, J. (2012). Predictive mental workload modeling: implications for system design. *Journal of Systems Engineering, 15*(4), 448-460.

DAU. (2013, Sept 16). *Defense Acquisisiton Guidebook.* DAU. Retrieved from Defanse Acquisition University.

Defense, U. S. (2018). *National Defense Strategy.* Washington, DC: USDOD.

Durscher, R. (2019). *pyCAPS: A Python Interface to the Computational Aircraft Prototype Syntheses.* San Diego, CA: AIAA.

Gillespie, S. (2017). System Of Systems Architecture Feasibility Analysis to Support Tradespace Exploration. *12th System of Systems Engineering Conference* (p. 6). Waikoloa, HI: IEEE.

Haimes, R. (2013). *The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry.*

Hummel, J. (2016). *Model Based Conceptual Design (MBCD) Modeling Your Concepts.* MBSE Solutions.

Hunter, S. (2015). *Validating Model-Based Design Simulation: The Impact of Abstraction and Fidelity Levels.* IEEE.

IEEE. (2000). What Is Systems Engineering. *IEE Aerospace and Electroinic Systems Magazine Jubilee Issue*, 2.

Lucas, T. (2002). *When is Model Complexity Too Much? Illustrating the Benefits of Simple Models With Hughes' Salvo Equations.* Naval Research Logistics.

ODASD(SE). (2018, June). Digital Engineering Strategy. Washington, D.C., United States.

OMG. (2022, Feb). *What is SysML?* Retrieved from OMG Systems Modeling Language: https://www.omgsysml.org/what-is-sysml.htm

Roper, W. (2020, Sept 15). Take the Red Pill: The New Digital Acquisition Reality. Washington, DC, United States.

Tarkian, M. (2009). *Design Reuse and Automation on High Level CAD Modeling for Multidisciplinary Design and Optimization.* Linkoping University.

University, D. A. (2020). *Defense Acquisition Guidebook.*

Ward, A. (2020, Dec 15). *How Long Does it Take to Learn a New Language?* Retrieved from https://www.rypeapp.com/blog/how-long-does-it-take-to-learn-a-new-language/

Warniez, a. (2017). SysML Geometrical profile development for physical. *14th Mechatronics Forum International* (p. 7). Karlstadt, Sweden: HAL Open Science.

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 074-0188*
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE *(DD-MM-YYYY)* 18 -02-2022	**2. REPORT TYPE** Master's Thesis		August 2020 – March 2022
TITLE AND SUBTITLE GEOMETRIC GENERATION THROUGH MERGING OF SYSML AND ENGINEERING SKETCH PAD		**5a. CONTRACT NUMBER**	
		5b. GRANT NUMBER	
		5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Miesle, Alexander J., Captain, USAF		**5d. PROJECT NUMBER**	
		5e. TASK NUMBER	
		5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-7765			**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENV-MS-22-M-237
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			**10. SPONSOR/MONITOR'S ACRONYM(S)** **AFRL/RHIQ (example)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

The United Stated Department of Defense (DoD) and Air Force (USAF) have placed increased emphasis on the utilization of modern systems engineering (SE) practices within the current and future acquisitions lifecycle. This call is driven by the current rate at which near-peer adversaries such as Russia and China are increasing their defense system capabilities and catching up or surpassing the Unites States in certain operational regions. To aid in the transition to a Digital Engineering and Digital Twin dominated acquisitions process, this thesis presents a method with which SysML and geometric tools can be linked within both new and existing models built through MBSE practices. Specifically, this thesis focuses on the use of Cameo Systems Modeler and Engineering Sketch Pad to explore the link between Systems Models and Geometric Models. These tools, in conjunction with a well-developed pattern were exercised using a simple, ground-up approach model and an existing model. The result of this thesis is a stepping-stone to more complex geometric generation and a direct pipeline from SysML to analysis tools which require either solid models or volume meshes (i.e., CFD, FEM, RCS calculations).

15. SUBJECT TERMS
SysML, MBSE, Geometry Modeling, Digital Engineering, Digital Twin

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Ford, Thomas, AFIT/ENV
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	63	**19b. TELEPHONE NUMBER** *(Include area code)* thomas.ford@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18