WORCESTER POLYTECHNIC INSTITUTE

ELECTRICAL AND COMPUTER ENGINEERING PROGRAM

Automate Model Data into a Physics-Based Analytic Software

A Major Qualifying Project

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the
Degree of Bachelor of Science by:

Maya Ellis

Project Advisors:

Professor Shamsnaz V. Bhada

Worcester Polytechnic Institute

John J. Roberts

Alex Mouw

MITRE

Sponsored By:

The MITRE Corporation

October 12, 2022

Acknowledgements

Thank you to Professor Shamsnaz V. Bhada, John J. Roberts, and Alex Mouw for advising this project and ensuring its success. Additional thanks to MITRE's Taz Biesinger and Jim Dalton. This project would not have been possible without all your contributed time and expertise. Thank you for the opportunity to work on this project and learn from it.

Table of Contents

Table of Figures

Table of Tables

Abstract

Digital engineering (DE), an emerging aspect of systems engineering, has the potential to continue to increase efficiency and improve the quality and performance of both commercial and military systems. As technology and the practice of DE evolve, previously used modeling processes need to be adapted to enable effective DE-based design and development. Digitizing engineering artifacts means end to end integration of all domain-specific modeling tools with each other and with Systems Modeling Language (SysML). There is a lack of integration between SysML architecture models containing system design details and physics-based models of the deployed system in operational use to understand missions impacts of proposed design. If fully integrated these tools could show performance differences due to design changes quickly and iteratively. In this research, a mission scenario was simulated to understand how to implement an end-to-end integration between SysML using Cameo tool and a physics-based simulator, Systems ToolKit, to demonstrate how design changes to two types of system can be shown to produce different operational capabilities long before the systems are fully designed and physically constructed.

1. Introduction

Systems engineering (SE) and digital engineering (DE) processes are heavily utilized by commercial industry as well as for DoD programs. Model based systems engineering (MBSE) uses digital models to represent systems engineering artifacts which allows for efficient design and analysis performance compared to previous paper document-centered systems engineering processes [1].

The MITRE Corporation has been pioneering the evolution of a DE platform. They have provided both the initial DE modeling problem statement and resources to permit this project to be executed. MITRE began in 1958 as a private, not-for-profit company to provide engineering and technical guidance for the United States Air Force. This served as the foundation for the growth of federally funded research and development centers (FFRDC) managed by MITRE. Today, MITRE operates six of 42 FFRDCs in existence [2]. MITRE addresses complex national challenges that threaten the United States' safety, security, and prosperity.

A current gap related to military systems modeling is the lack of integration between Systems Modeling Language (SysML) architecture models containing system design details and physics-based models of the deployed system in operational use to understand mission impacts of proposed design changes quickly and iteratively. This project focused on prototyping and demonstrating a bridge for that gap.

This project's Python implementation of an end-to-end integration between a system design captured SysML in Cameo and its operational deployment, as implemented in Systems ToolKit (STK), could be used for future modeling and analysis. This integration will allow for smoother and more efficient information distribution between industry developers and

operational users of systems. Additionally, with a successful integration between the applications, future work can focus on furthering automation between the two enabling increased productivity.

2. Background

For the United States Department of Defense (DoD), the heightened use of modeling, simulation, and analysis (MS&A)'s as part of DE enables the transition of military system acquisition from paper document-intensive to computer based. Although, as modeling technology progresses, new barriers can be introduced. A challenge related to MBSE is the current lack of integration between SysML-based models, such as Cameo, and physics-based models such as Systems ToolKit (STK). STK and other physics-based models are primarily used to understand how the systems under development will perform in realistic operational scenarios. Defense contractors design the proposed "to-be-built" systems modeled through SysML. Without integration of design models and real-world scenario testing, not all stakeholders will be able to efficiently access the critical system information and assess the quality and expected performance of a system design. This project utilized a fictitious operating environment of an internal Continental United States operation (to avoid possible classification issues by introducing real world scenarios) with changing system design parameters to research and implement the end-to-end integration capability between SysML in Cameo and STK through Python. To provide background for the topics of this project, this chapter describes:

- Systems Engineering
- Digital Engineering
- SysML Architecture Models
- Physics-based Models
- Current Challenges and Limitations

2.1 System Engineering

According to the International Council on Systems Engineering, INCOSE, “systems engineering is a transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using system principles and concepts, and scientific, technological, and management methods” [3]. The concept focuses on the entire technical effort to ensure all parts can function correctly in the proposed operational environment. The Government Accountability Office (GAO) research shows that programs with early detailed systems engineering before development began had better project outcomes [4]. Until recently, systems engineering was conducted through a documented-intensive process. What used to be writing out documentation for requirements, technical reviews, and architecture design transitioned to models in a digital engineering environment [5]. Although systems engineering has transitioned to models it still lacks integration with domain specific digital tools such as STK for physics-based models.

2.2 Digital Engineering

Digital Engineering, the fourth industrial revolution, is evolving the work in the defense industry at a rapid rate [6]. Figure 1 illustrates a timeline of the sequential industrial revolutions. In June 2018, the U.S. Under Secretary of Defense for Research and Development released the U.S. Department of Defense (DoD) Digital Engineering Strategy (DES). The DES anticipates that with the rise of more complex systems, such as in the realm of new weapon systems for the DoD, digital engineering (DE) will lead to greater efficiency and improved quality of systems. The crux of digital engineering is the creation of computer readable models to represent all

aspects of the system and to support all the activities for the design, development, manufacture, and operation of the system throughout its lifecycle [5].

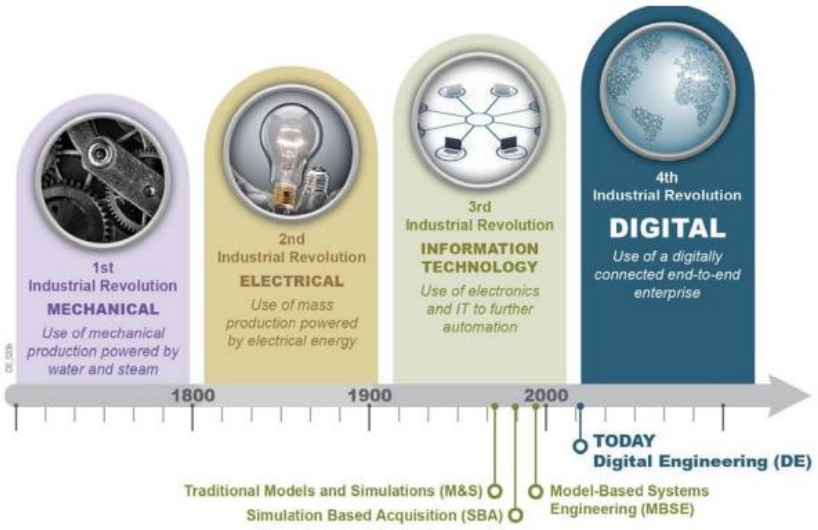

Figure 1: Timeline of Industrial Revolution [19]

Many engineering disciplines have adopted digital engineering approaches, such as aerospace and mechanical engineers utilizing computer aided design tools (CAD) instead of drafting boards. Another example is software engineers using software GUIs and one single environment for writing, checking, compiling, and running code instead of their previously used text editors. Similarly, systems engineers have adopted model-based system engineering (MBSE) to perform analysis and design tasks more efficiently than ever before [7]. These models or digital twins provide a digital representation of a system, mission, or process that is comprised of transdisciplinary models and simulations [6].

2.3 SysML Architecture Models

SysML architecture models are used for general-purpose systems architecture modeling. They support specification, analysis, design, verification, and validation of a broad range of

systems [8]. One of these tools is Cameo Enterprise Architecture (referred to as simply Cameo). Other SysML-based architecture tools include Rhapsody, Visual Paradigm, and Microsoft Visio. Cameo was selected for this project because of prior experience with the application. It fully supports all architectural framework products ensuring necessary results. Cameo is used widely within MITRE and worked well for this project's research goals.

Cameo

The Cameo Enterprise Architecture tool is developed CATIA No Magic. It is primarily used to measure and visualize architecture to improve project results. Cameo conveys knowledge of, well represents and communicates complex architecture, reducing assumption, misconception, and system risks [9].

2.4 Physics-based Models

Physics-based models and simulations are constrained by the laws of physics to provide virtual environments to simulate the experience of what could be in the real world [10]. Examples of physics-based models and simulations include BEM, which simulates building energy use and COMSOL Multiphysics, which simulates device designs and processes.

Government contractors use various modeling tools within collaborative, DE-based weapon system design and development environments for the simulation of proposed systems. One of these physics-based models is Systems ToolKit (STK). Other physics-based models and modeling frameworks used to analyze mission effectiveness of DoD systems include EADSIM, Brawler, and AFSIM.

STK was selected for this project because STK can model all the assets involved in complex, critical systems such as communications, radar, coverage, and can visually report

results. This application allows users to see what impact all system elements have on the entire mission.

Systems Tool Kit

Systems ToolKit is developed by Ansys. It is primarily used in the domains of aerospace, defense, telecommunications, and other industries. It utilizes a physics-based modeling environment to permit analysis of platforms and payloads in a realistic mission context. A user is able to model systems inside a realistic, time-dynamic, three-dimensional scenario that includes high resolution terrain, imagery, and radio-frequency environments. The application creates the ability to simulate an entire system-of-systems at any location and time to gain an understanding of mission performance and behavior [11].

2.4 Current Challenges and Limitations

One challenge identified by MITRE in the MBSE domain is the lack of integration of MBSE tools with physics-based models. Although MBSE supports the system engineering activities of requirements, architecture, design, verification, and validation, these efforts need to be connected to physics-based models to be used to understand design change impacts on mission effectiveness. Without proper representation of a system's data, not all stakeholders will be able to access the necessary information [5]. As the sponsor of this project, MITRE has been pioneering the evolution of a Digital Engineering (DE) Platform to enable more efficient and effective design, development, and deployment of systems to enhance national security for the DoD. MITRE analysts use MBSE tools to understand overall mission effectiveness implications of design changes. Within current DE environment instances based on this platform, analysts do not yet have an end-to-end capability that automates the ingestion of updates to system design

data, as contained in MBSE tools, into physics-based mission simulation tools. This integration would permit continuous examination of system behaviors when faced with various operational threat scenarios. The integration between MBSE and physics-based models can improve predictive, prescriptive, and cognitive capabilities of military systems [12].

3. Operational Scenario Description

The Department of Defense (DoD) is America's largest government agency. Their mission is to provide the military forces needed to deter war and ensure the United States' national security. Components of the DoD include the Army, the Marine Corps, the Navy, the Air Force, Space Force, Coast Guard, and National Guard [13]. This project will utilize a hypothetical Air Force example, focusing on generic airborne command and control aircraft and surface-to-surface missiles as the example systems under study for potential next generation enhancements using DE processes, enabled by modeling and simulation.

3.1 Operational Scenario

The operational scenario for this project, shown in figure 2, was a Continental United States (CONUS) only fictitious situation to avoid any potential security issues that could result from using potential adversary countries and real system performance parameters. In this scenario, the CONUS is separated down the middle into "RedLand" (Adversary Territory) and "BlueLand" (Friendly Territory). The BlueLand's priority is upgrading key systems to conduct future strikes on the RedLand if needed to respond to adversary actions. Within the figure, there are command and control (C2) Aircrafts and surface-to-surface missiles (SSMs), both newly funded systems in this fictitious example. Since they are assumed to be newly funded, analysis needs to be performed continually during the design and development cycles to characterize the operational benefits to design implementation choices.

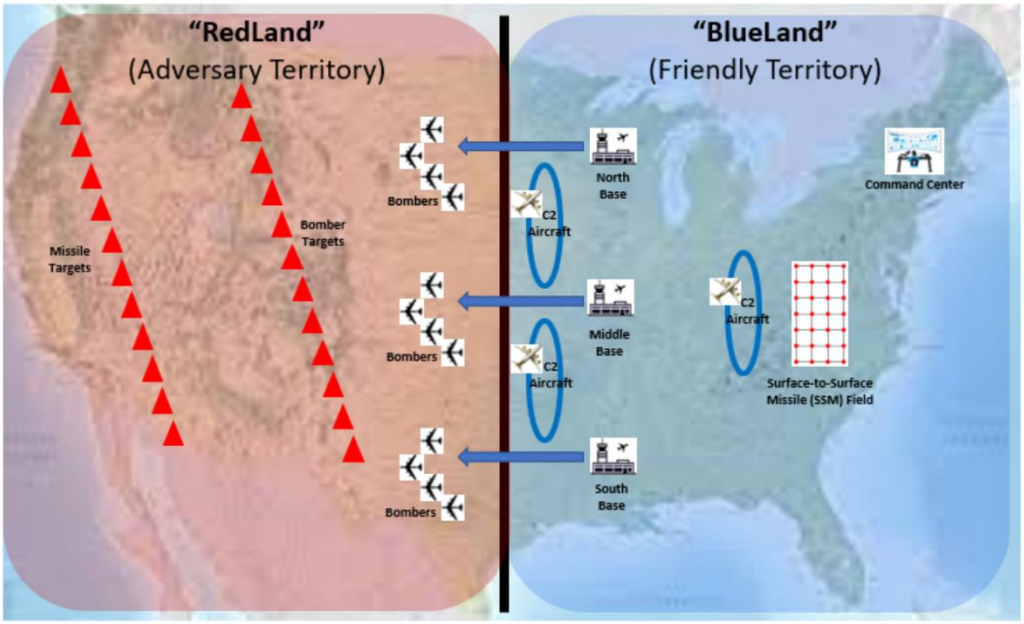

Figure 2: Fictious Operational Scenario to be Analyzed

3.2 Command and Control Aircrafts

Command and control (C2) as a concept is a means toward creating value, such as accomplishing a mission. It's the effort of multiple individuals, organizations, and resources to achieve the same end goal [14]. An example of an airborne C2 system is the E-3 Sentry, known as the Airborne Warning and Control System (AWACS). The E-3 is an airborne C2, management, surveillance, target detection, and tracking platform. The E-3 radar has a range of more than 250 miles (375.5km), its max speed is 360mph, and its engine range is more than 5,000 nautical miles (9,250km) [15]. We will use and vary generic parameters such as these in the investigation of the redesign of a hypothetical platform of this type.

Figure 3: E-3 Sentry (AWACS) [20]

3.3 Surface-to-Surface Missiles

There are multiple kinds of surface launched missiles such as surface-to-air missiles (SAMs) and surface-to-surface missiles (SSMs). This project will examine the redesign of SSMs. SSMs are missiles designed to be launched from the ground or sea with a strike target on land or sea. This differs from SAMs, which launch from the ground, but target airborne objects such as an aircraft [16]. For the purpose of this project scenario in creating an end-to-end integration, the SSMs used will not be ICBM because the United States, from a latitude perspective is less than 5500km. Typically, ICBMs have a range greater than 5500km. Therefore intermediate-range ballistic missiles (IRBMs) will be utilized as they have a range between 3500km and 5500km. This project will use and vary generic parameters such as these in the investigation of a hypothetical missile of this type.

4. Research Description

Digital Engineering is becoming increasingly essential for more efficient and effective design, development, and deployment of systems to enhance the national security for the Department of Defense (DoD) [13]. Although, a challenge that affects the growth of digital engineering and MBSE in the DoD setting is the gap of integration between MBSE tools and physics-based models, which are utilized by the contractors designing and building these proposed systems. The language primarily used in MBSE environments is Systems Modeling Language (SysML). Currently, SysML does not have a bi-directional interface with Physics-based modeling tools such as STK. The advantages of integrating SysML modeling tools such as Cameo with STK include automating the capabilities of both tools without the need to switch between applications, extending the types of analysis needed to perform mission capabilities as Cameo lacks what STK has, and visualizing a 3D scenario to interpret SysML data, another capability Cameo does not have.

This project simulated a fictitious operational scenario in Cameo utilizing SysML to research how to implement an end-to-end integration capability between SysML and STK through Python. To limit the scope of this project, the parameters selected were made for simplified system designs. In a real-world simulation, the scenario would be more complex. The system design parameters utilized for this project included launch performances for the missile. The aircraft parameters varied included the engines and external communications performance. Through this research and implementation, data was obtained to make informed conclusions. Overall, the research question is: **How can a bi-directional interface between Cameo and a physics-based modeling platform such as STK be achieved?**

This question was answered by:

- Implementing a python script that integrates data one way from SysML to a physics-based model. This is Hypothesis 1, which tested data flow accuracy for all known variables from Cameo to STK.
- Improving the baseline python script to integrate data bi-directionally. This is Hypothesis 2, which tested the data flow accuracy for all known variables from Cameo to STK and STK to Cameo.

Hypothesis 1: Data from SysML in Cameo to physics-based models such as STK can be integrated one way by a Python script.

The first goal of this project was to get the data from the system designs in Cameo to be readable and used within STK to create data for design trade analyses. The two ideas on how to go about doing the initial integration were to export the data via XML or CSV. Both exports had their pros and cons and whatever became most accessible in the process became the files of choice.

Hypothesis 2: Data from SysML in Cameo to STK can be integrated by a Python script to automate the flow bi-directionally.

Once the initial integration was complete, the next step was to create a foundation for extracting the data to be readable for a user to adapt the initial system design. The most efficient way to do so with the current Python script and resources was to write the results from STK to a CSV file. This file could then be interpreted by a user or could even be reuploaded to Cameo to change the initial system design values.

5. Integration

This section details the process for integrating Cameo and STK. Figure 4 displays the preliminary verification architecture in the form of a sequence diagram. The goal of the research was to thread the information from Cameo to STK and back. This system design parameters from Cameo would be exported to the Python Interface Script. The goal of this script was for the model information to be translated into data that STK could understand. This process requires the analyst to first develop and implement an operational scenario in STK. STK would then internally simulate the scenario for the data parameters given. After completion of the STK simulation runs, the information from STK would be sent back to the Python Interface Script. The Python Script would create a CSV file containing the STK information results. A CSV file is used to import and export data from various applications. It's a plain text file that stores tables and spreadsheet information. The user could utilize this information for supplemental information alongside STK graphical results in a design trade analysis. If and when subsequent changes are made to a system design, the analyst can then edit the Cameo model to initiate the process again.

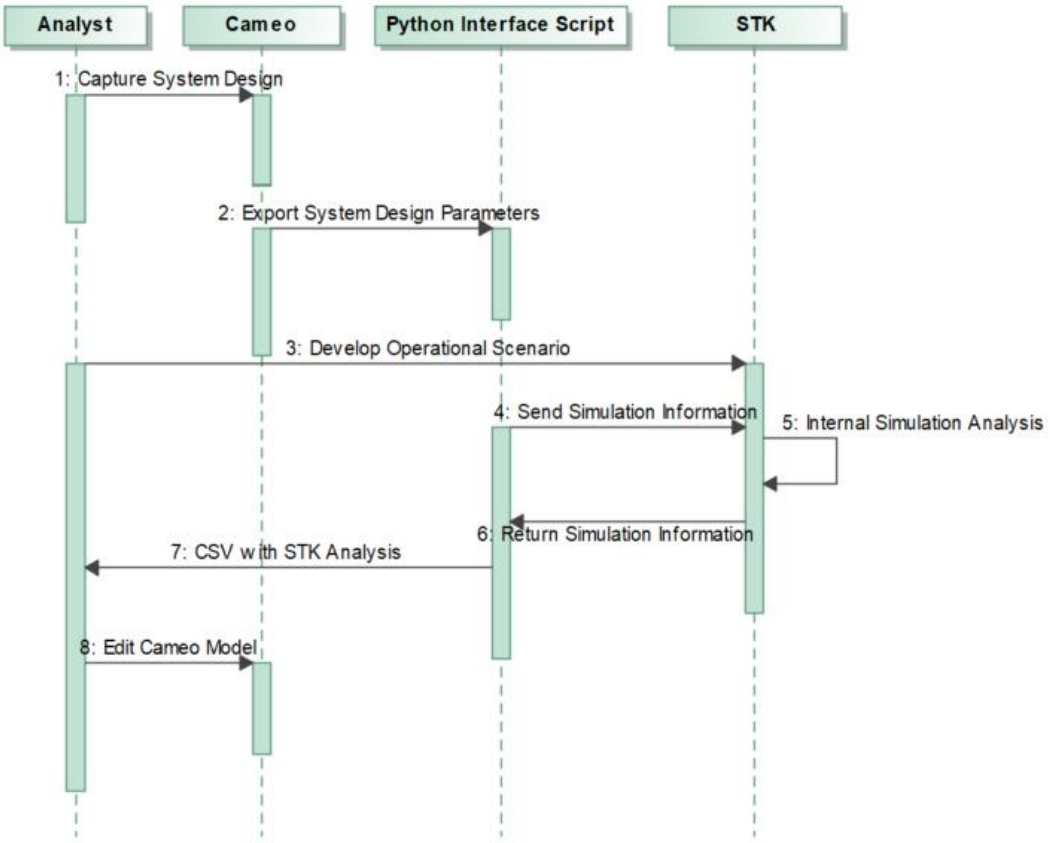

Figure 4: Preliminary Verification Architecture for Integrating Cameo and STK

5.1 Cameo Model

Figure 5 illustrates the aircraft block definition diagram (BDD) in Cameo. This diagram contains the example E3AWACS as the physical system being the aircraft. This simplified diagram of the system shows two of the key subsystems associated with it, the engines, and the external communications which include the antenna and transmitter. The engine's subsystem design includes properties such as the engine's maximum range, the engine's maximum altitude, and maximum speed. For this example, the engine maximum range parameter is set to 9,250km, the engine maximum altitude parameter is set to 9.15km, and the engine maximum speed is 0.2km/s. The antenna subsystem design information is split into several value properties:

frequency, physical length, efficiency, refraction atmosphere altitude, and knee bend factor. The transmitter subsystem design information is split into the following properties: frequency, power, data rate, polarization reference, and tilt.

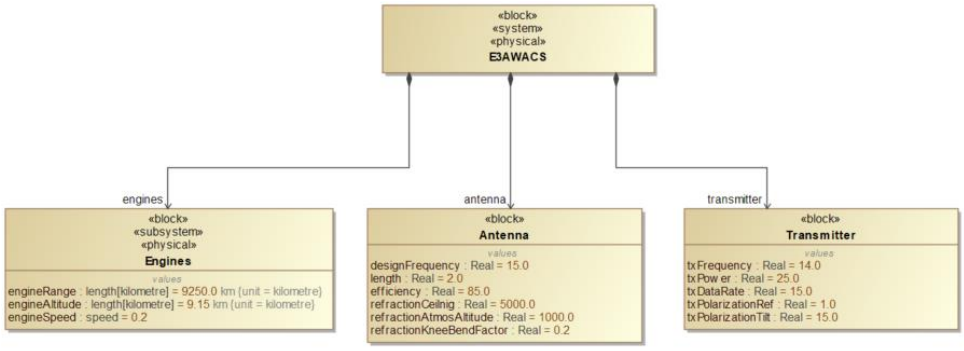

Figure 5: Cameo Block Definition Diagram of Aircraft

Figure 6 illustrates the missile BDD in Cameo. This diagram contains an example missile as the physical system. In this scenario, the system has one defined physical subsystem, the engines. The engines subsystem has a set value property of the missile altitude range, or how far the missile can physically travel upward. This altitude range parameter is currently set at 4,200km.

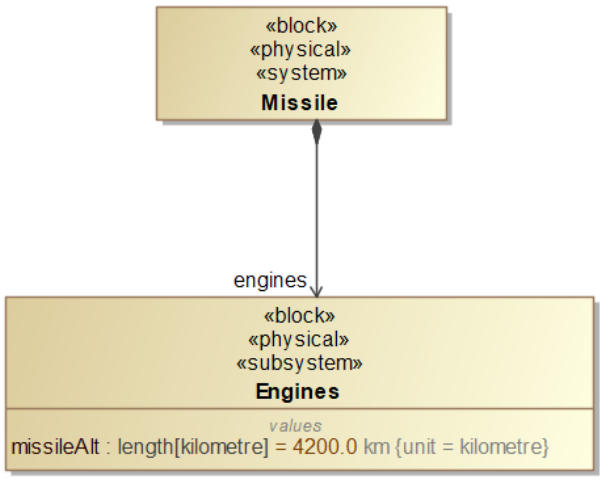

Figure 6: Cameo Block Definition Diagram of Missile

5.2 Exporting via XML

The initial attempt of exporting the information from Cameo was to export via an Extensible Markup Language (XML) file. The reason for choosing this export was because XML is a language that defines a set of document coding rules in a format that is both human-readable and machine-readable. Typically, there are both tags and text in the file, which are used to give the data in the structure.

Exporting XML from Cameo was not a straightforward process for this scenario. The Report Wizard portal was referred to as a solution to export via XML when looking through Cameo documentation. However, the export did not include any of the necessary data needed for the Python script as shown in figure 7. It may be worth looking further into XML and Cameo integration in the future. Nevertheless, to continue the integration process for this project in the necessary timeframe, another approach was taken.

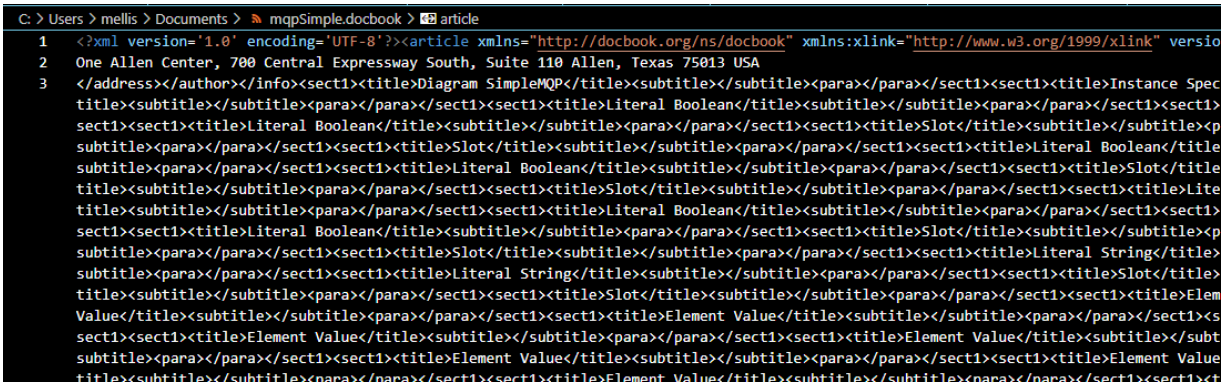

Figure 7: Output XML Failed Attempt

5.3 Exporting via CSV

The next attempt for exporting the information out of Cameo was to do so by a Comma Separated Values (CSV) file. Similarly, to XML, it's a format that stores structured data using text files. However, in a CSV file, each line is a data record. This made it easier to visually read the CSV file, and for Python to interpret the data than it was with an XML file.

The Aircraft and Missile BDDs were first expressed in a generic table within Cameo. The generic table allowed for a user to choose which element types are displayed in the table. Originally, the table showed the element types chosen for this scenario including the element name, owner, applied stereotype, part, attribute, and default value. Although these element types gave context for the scenario created, essential information from this table were the name of the element and the default value shown in table 1. This was because the values from the Cameo diagram would be used to input into the Python script for analysis.

The generic table from Cameo was then exported via a CSV file. This file held all the needed information and displayed it almost exactly to how the generic table looked in Cameo.

Value Property	Default Value
Missile Altitude	4200km
Engine Range	9250km
Engine Altitude	9.15km
Engine Speed	0.2km/s
Transmitter Data Rate	15Mb/s
Transmitter Frequency	14Ghz
Transmitter Polarization Ref	1
Transmitter Polarization Tilt	15 Degrees
Transmitter Power	25dBW
Antenna Refraction Knee Bend Factor	0.2
Antenna Refraction Atmosphere Alt.	1000m
Antenna Physical Length	2m
Antenna Efficiency	85%
Antenna Frequency	15Ghz
Antenna Refraction Ceiling	5000m

Table 1: Value Properties and their Default Values from Cameo

5.4 Native STK Information

The scenario information that could be found in STK was independent of the system designs in Cameo. It included the physical laydown of Blue Order of Battle locations, the strike plan including weapon assignments, the bomber capabilities, the threat systems (for this scenario, only including the RedLand targets), the weather, date, and time.

5.5 Creating the Python Script

The next step was creating the Python script that would be read by STK to execute simulation runs of the scenario in Cameo. To ensure that the CSV files could be read correctly via Python, Algorithm 1 (figure 8) was implemented. The SysML file was read with the CSV reader function in Python. Then, the data was put into a Pandas Data Frame. A Pandas Data Frame is a two-dimensional size-mutable, tabular data structure with labeled axes. The three principle components of the data frame consist of data, rows, and columns [17]. The purpose for this was to print all the data from the excel file, then put the necessary data (data names and

values) into a table. From there, the values of the properties were separated into their own strings with indexes to create the ability for the values to be inputted when needed to for various STK objects.

Algorithm 1: Read Output File from Cameo

Input: CSV of SysML Model
Output: SysML CSV Reader

```
SET data = SysML csv output file
data = list(csv.reader(data))
Create list for names
Create list for values
        for column in csv file do
                Append name list
                Append value list
        end for
SET names = series of names from list
SET values = series of values from list
SET dataframe = pd.concat([names, values], axis = 1)
print(dataframe)
```

Figure 8: Algorithm 1

Algorithm 2 (figure 9) shows the process for starting up the automation process of STK, taking in the information from the SysML CSV reader, and generating a scenario.

```
Algorithm 2: Startup STK
Input: SysML CSV Reader
Output: Generate Automated STK Scenario
   Function Startup_STK (Cameo filename)
   scenario initialization
   SET stk = STK Desktop Application
   get root object
   set data format
   SET stk root = new scenario
   SET scenario = root current scenario
   Import CSV data
   Create 10 Targets
            set coordinates
   Create Aircraft
            declare waypoints
            create antenna
            create transmitter
   Create Facility
            declare position
            create antenna
            create receiver
   Create 10 Missiles
            set to propagator ballistic
            declare launch coordinates
            declare impact coordinates
```

Figure 9: Algorithm 2

Algorithm 3 (figure 10) computes the access of the STK objects; the access was dependent on their property settings from Cameo.

```
Algorithm 3: Compute Access and Extract Data Providers
Input: STK Objects
Output: STK Access Computations
   #Compute Access
   SET access =
"STKOject".GetAccessToObject("STKObject")
   access.ComputeAccess()
   #Extract Data Providers
   SET accessDP =
access.DataProviders.Item('Access Data')
   access necessary data here
   print(accessed data)
```

Figure 10: Algorithm 3

Finally, Algorithm 4 (figure 11) took the accessed data from STK and passed it through a panda data frame. This data frame was sent to a CSV file for supplemental data for design trade analysis.

Algorithm 4: Export Extracted Data
Input: STK Access Computations
Output: CSV of Accessed Data

```
SET x = series(accessed STK data)
SET y = series(accessed STK data)
SET z = series(accessed STK data)
SET dataframe = pd.concat([x, y, z], axis = 1)
print(dataframe)
SET csv_data = dataframe.to_csv("output file path")
```

Figure 11: Algorithm 4

The result of the integration process was a simple automation of data flow between Cameo and STK. This created a foundation of a thread that can be built upon for future use. This attempt allowed a Python Script to ingest CSV data from SysML in Cameo and pass it through to STK to create a simulation analysis. A user could use this information from the analysis to change the initial system design data in the Cameo model.

6. Analysis Methodology

The integration between Cameo and STK using Python provides a new linkage that was previously nonexistent. With this link, varying system designs were created to compare the measure of effectiveness between different sets of runs of the fictitious scenarios. The analysis methodology followed the flow shown in figure 12. First, the initial design was created in Cameo, then it was interpreted through the Python script and placed into the operational scenario as implemented in STK. The simulation was then run, and the access values were extracted from the tool. The second step repeated the process, but with modified system design information. Finally, the results of the initial design runs were compared to the results of the modified design, this created the basis for an analysis of various designs. The initial design values that were changed were the missile altitude and aircraft maximum speed to determine if the analysis was accurate.

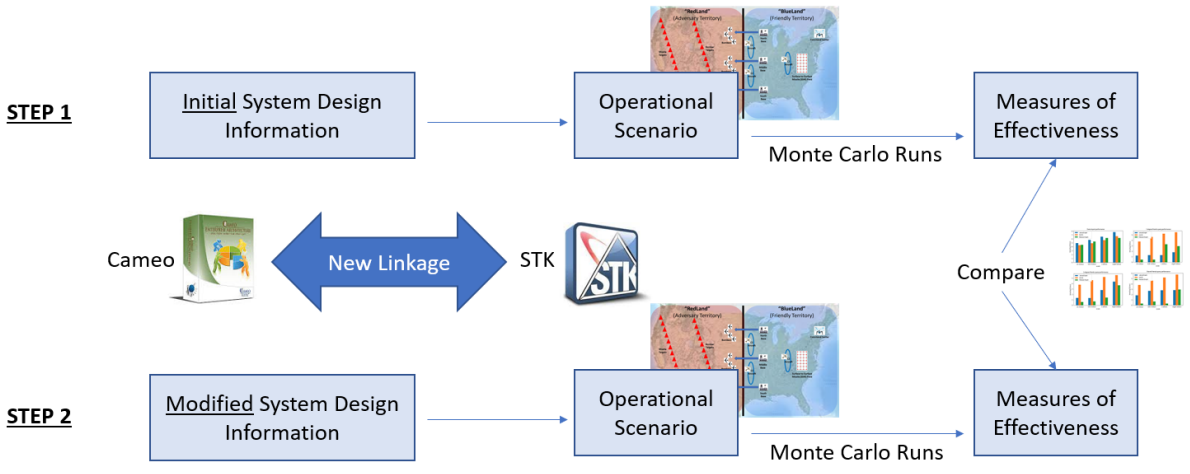

Figure 12: Flow of Analysis Methodology

7. Results

This section discusses the results from the design trade analysis of the initial scenario and the modified scenario. The results of the figures and tables labeled “Scenario 1” refer to the initial system design first modeled in Cameo, similarly, figures and tables labeled “Scenario 2” refer to the modified system design in Cameo.

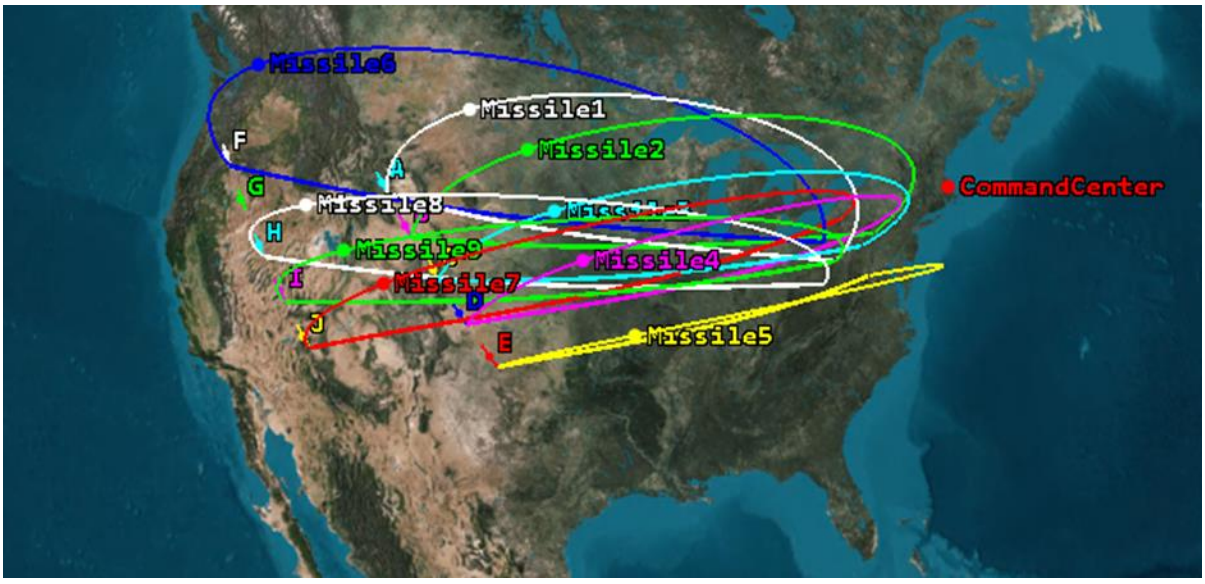

Figure 13: Missile and Target Layout in STK

7.1 Missile Design Results

The initial missile design had a maximum engine altitude range of 4200km, the modified missile design had a maximum engine altitude range of 1000km. To simplify the analysis, only one missile and one area target were viewed. The analysis reports generated from STK included the Altitude vs. Ground Range, the LLA Position, and a table of the Ground Range, Altitude, and Time next to each other to compare. The altitude and ground ranges were taken out of STK to validate that the proper maximum altitude was input from the Cameo scenario. In both cases this

can be seen as true as the highest altitude shown in figure 14 of scenario 1 is 4200km and the highest altitude shown in figure 16 of scenario 2 is 1000km.

The LLA Position Analysis graphs shown in figures 15 and 17 provided the missile angle (latitude and longitude) and the altitude over a function of time. This perspective gave the measure of effectiveness of the two missile designs. It can be seen in the first scenario that the missile takes approximately 48 minutes to impact its target location. This is significantly longer than scenario 2's missile, which only takes approximately 18 minutes. The analysis between the missile system designs showed that a shorter maximum altitude within a missile engine allows for shorter flight and therefore quicker impact at the desired location.

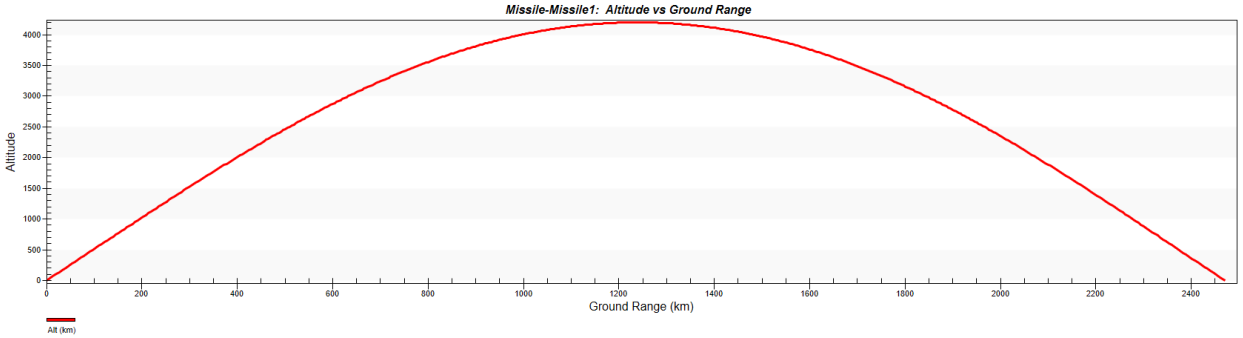

Figure 14: Scenario 1: Altitude vs. Ground Range, Max Altitude = 4200km

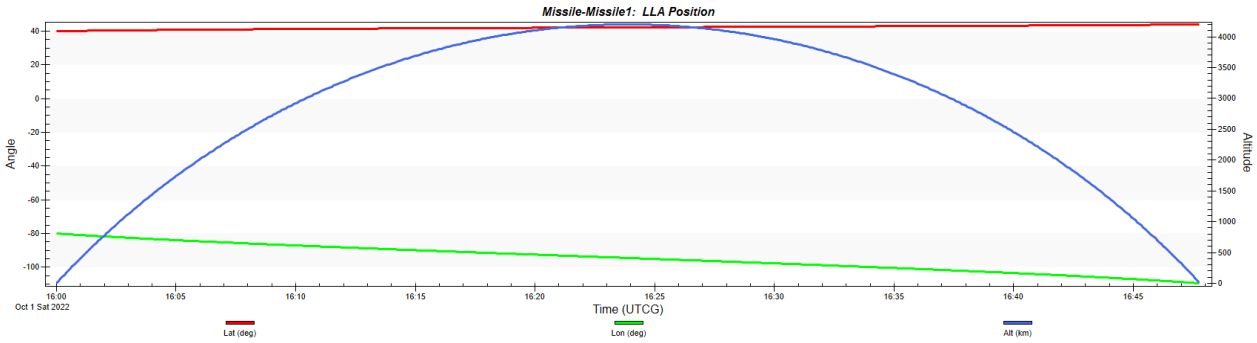

Figure 15: Scenario 1: LLA Position, Max Altitude = 4200km

	Ground Range (km)	Altitude (km)	Time (sec)
0	0	0	0
1	78.95629422	403.5340009	60
2	151.5996594	776.2207818	120
3	219.2745992	1121.183935	180
4	282.9551015	1440.909112	240
5	343.3685098	1737.412467	300
…	…	…	…
45	2267.026904	1050.730767	2700
46	2334.980571	700.1549283	2760
47	2408.104768	321.2575055	2820
48	2470.415069	-7.78E-10	2867.367537

Table 2 Scenario 1: Ground Range, Altitude, Time, Max Altitude = 4200km

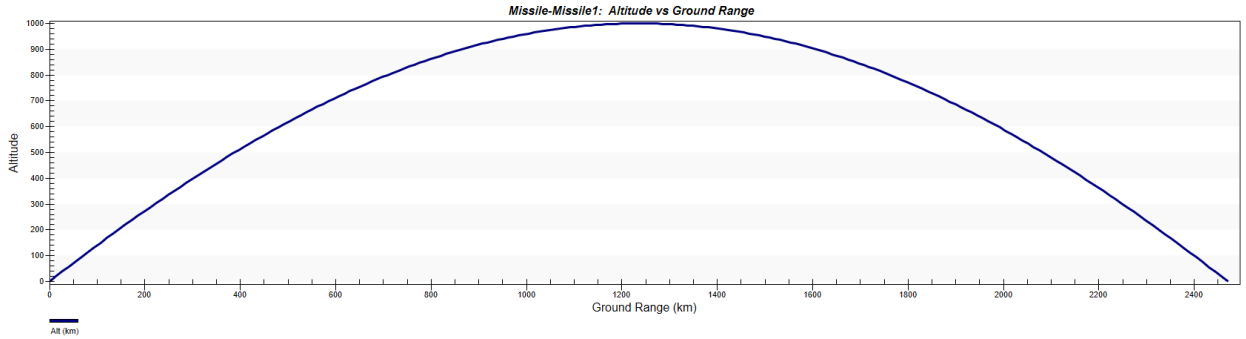

Figure 16: Scenario 2: Altitude vs. Ground Range, Max Altitude = 1000km

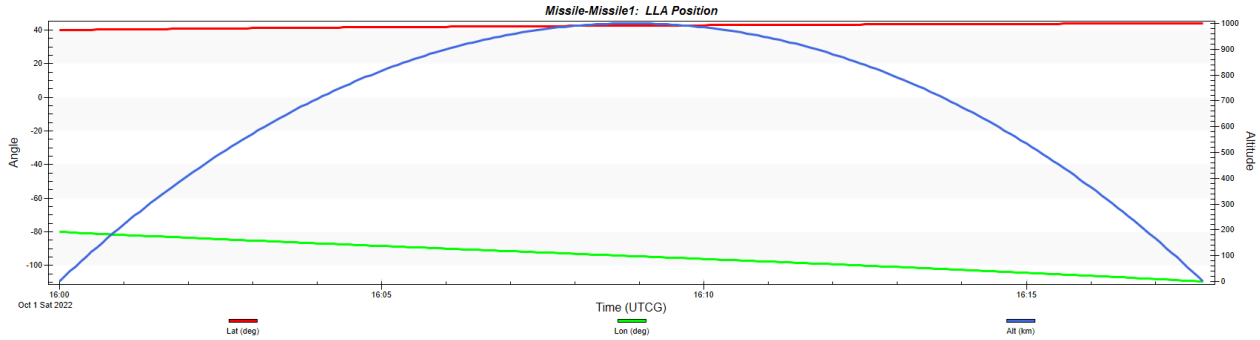

Figure 17: Scenario 2: LLA Position, Max Altitude = 1000km

	Ground Range (km)	Altitude (km)	Time (sec)
0	0	0	0
1	159.9128855	220.6665997	60
2	311.4045463	411.2124209	120
3	456.2670009	573.2337775	180
4	595.9293872	707.9901634	240
5	731.5667916	816.4665884	300
…	…	…	…
15	2051.442491	533.9185117	900
16	2197.766944	364.6373001	960
17	2351.137093	166.4517772	1020
18	2470.415069	-3.46E-11	1064.451801

Table 3: Scenario 2: Ground Range, Altitude, Time, Max Altitude = 1000km

7.2 Aircraft Design Results

To test the design of the Cameo modeled aircraft, access between the North, Middle, and South Facility antennas in and the aircraft transmitters were analyzed. The idea behind the initial scenario and modified scenario was to test when the aircraft was out of communication access. Communication access and callback time is important in real world missions. If an aircraft flies out of bounds of the access point, the facilities would be unable to give further instructions or even abort a mission.

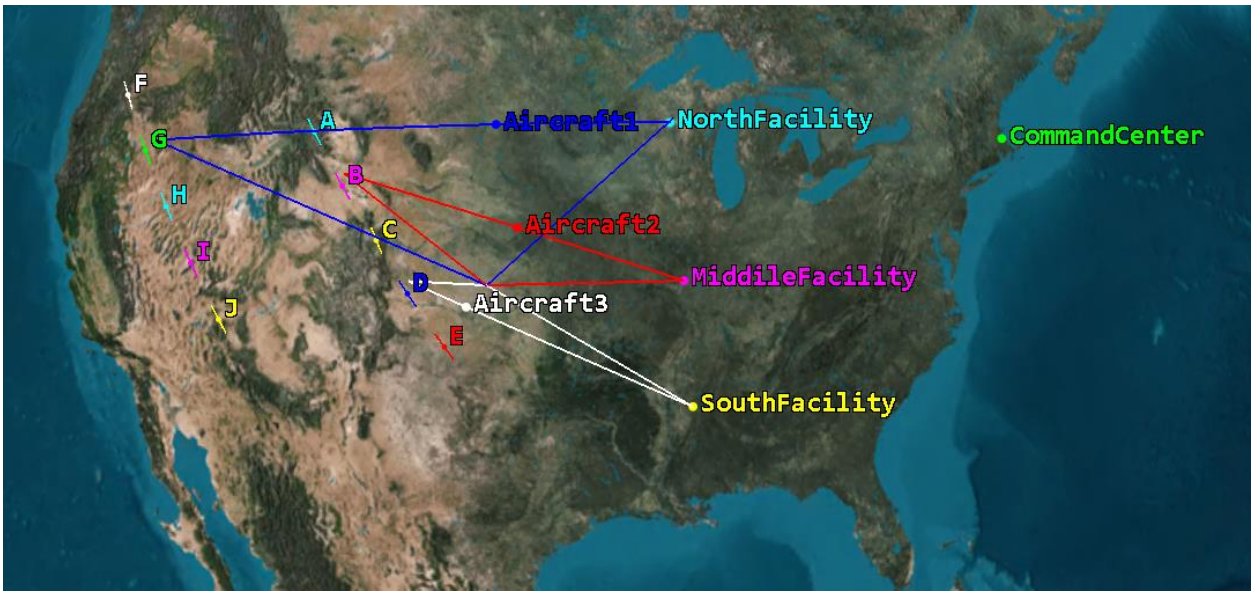

Figure 18: Aircraft Waypoints in STK

The initial scenario tested the aircraft transmitter parameters as modeled and followed simple waypoints. As shown in figure 18, the aircrafts initially flew from their respective facility centers to area targets “G”, “B”, and “D” and then flew back to the facilities. When this was analyzed, it was shown that the communication access was lost a little over an hour of flying to the target as shown in Table 4. Figures 19-21 show the facility and aircraft communication access (or lack of). The red lines in the graphics display the times in which there was access

between the two objects. The blank space displays no valid access between the facility and aircraft. With these results, scenario 2 aimed to create better communication access. Maximizing the amount of time a facility has communication access to an aircraft can increase the overall callback time in a given scenario.

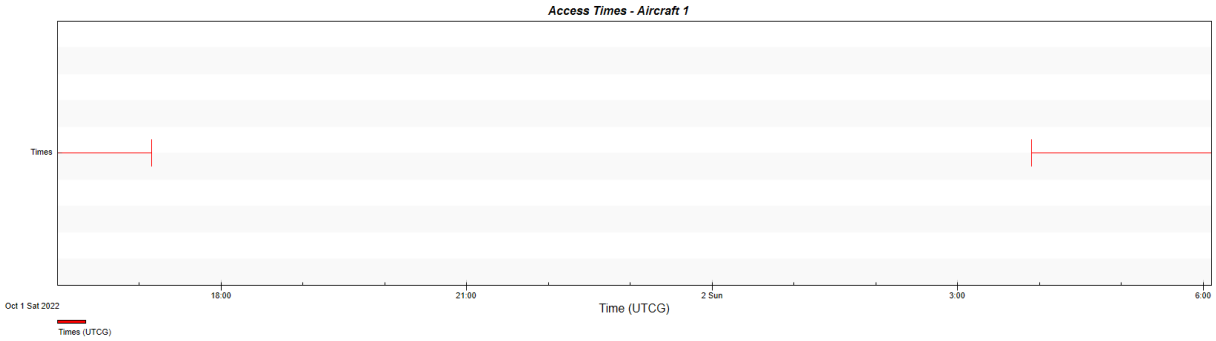

Figure 19:Scenario 1 Facility – Aircraft 1 Communication Access

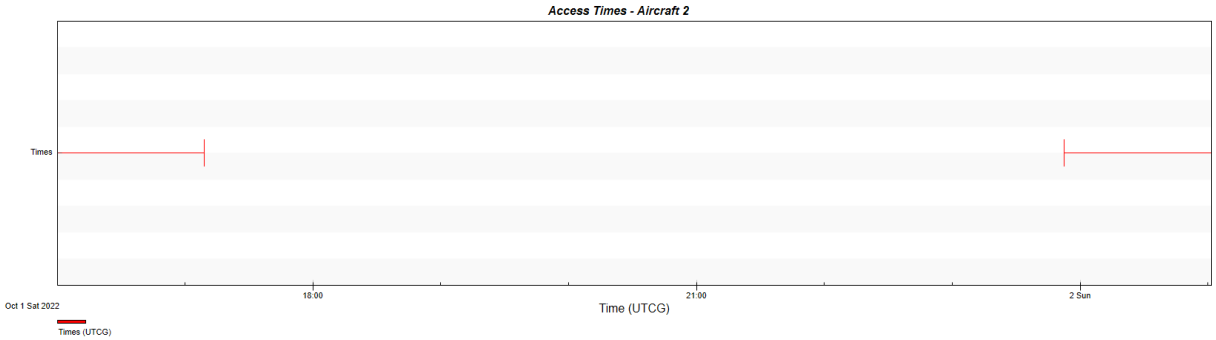

Figure 20: Scenario 1 Facility – Aircraft 2 Communication Access

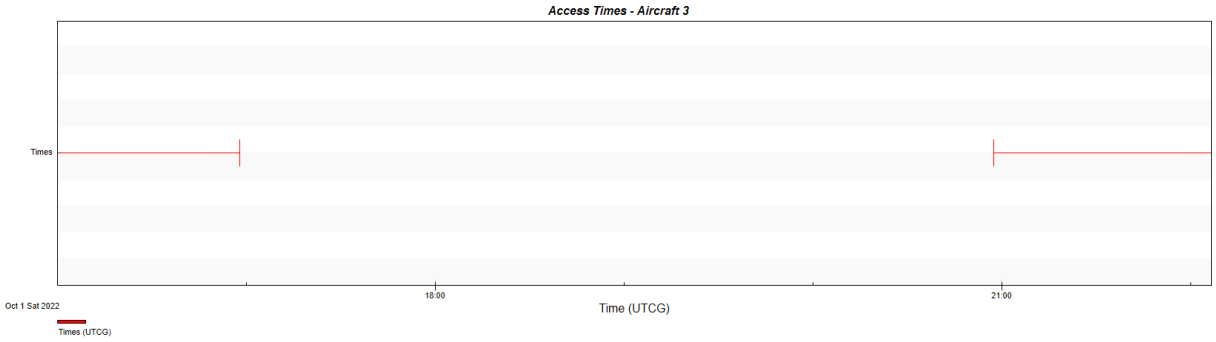

Figure 21: Scenario 1 Facility – Aircraft 3 Communication Access

Aircraft	Initial Callback Time
North Aircraft (1)	69.05 minutes
Middle Aircraft (2)	68.98 minutes
South Aircraft (3)	57.93 minutes

Table 4: 1 Scenario 1 Facility – Overall Aircraft Callback Time

In scenario 2, a couple options were explored for elongating the communication access between the aircrafts and their respective launch facilities. The first attempt was to decrease the speed at which the aircraft were hitting the waypoints. As shown in table 5, the slower the aircraft was going, the more time the facility had to callback and abort the mission. Although this was an effective method for this scenario, this would not be ideal in a time sensitive mission. Due to time constraints more design scenarios were not completed, however this set the foundation for communication access trade analysis. Furter research for this scenario to understand the communication systems within the fictitious aircraft in STK could look at the antenna, transmitter, and radar sensing. This could help evaluate a better idea of how communication parts impact overall aircraft access.

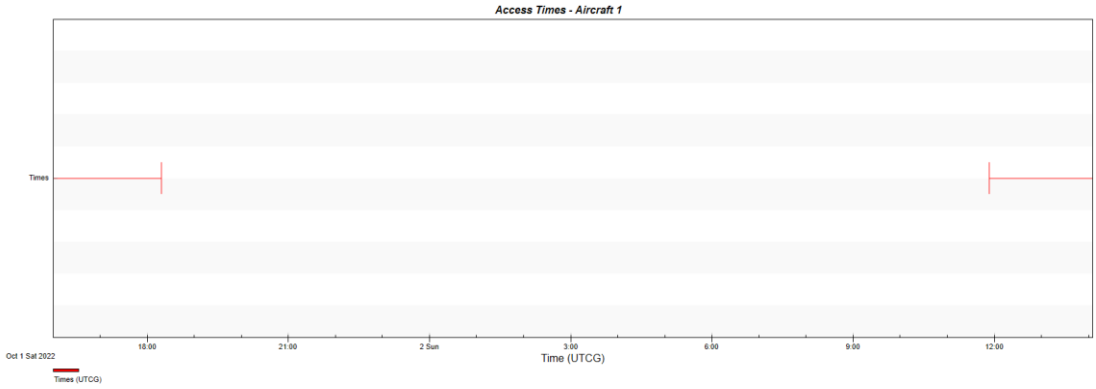

Figure 22: Scenario 2 Facility – Aircraft 1 Communication Access (slower speed)

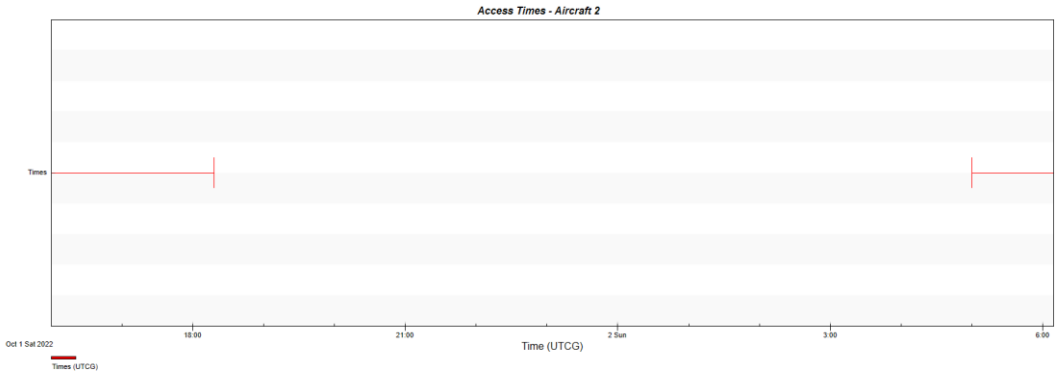

Figure 23: Scenario 2 Facility – Aircraft 2 Communication Access (slower speed)

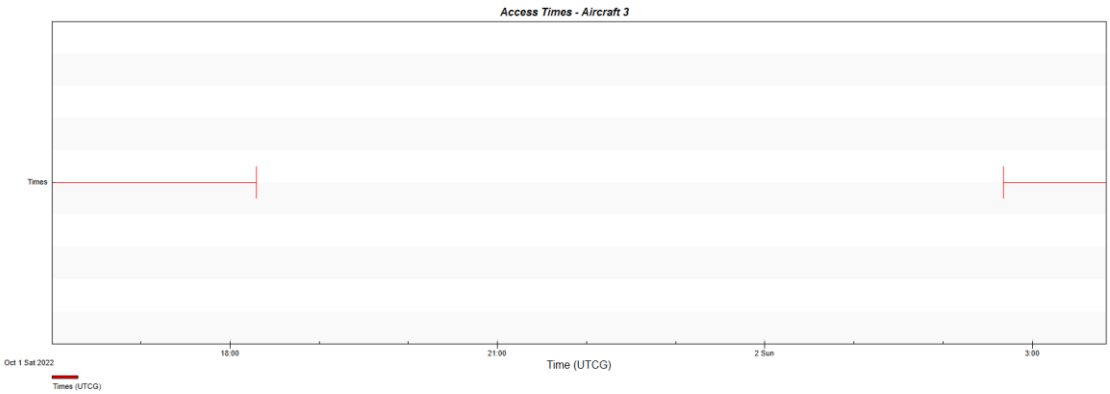

Figure 24:Scenario 2 Facility – Aircraft 3 Communication Access (slower speed)

Aircraft	Callback Time
North Aircraft (1)	138.1 minutes
Middle Aircraft (2)	137.9 minutes
South Aircraft (3)	137.9 minutes

Table 5: Scenario 2 Facility – Overall Aircraft Callback Time (slower speed)

This section highlighted the results of running the initial design simulation and a modified scenario after the analyzed data from STK was outputted via CSV files and graphs. These scenarios were used to understand the accuracy of the integration process. The missile design showed that the higher the missiles maximum altitude is, the longer it took for the missile to impact the target area. The second missile design showed that changing the missile altitude from 4200km to 1000km showed that the missile took significantly less time to reach its impact area. For the aircraft design, the goal was to create more access time between the facility and the aircraft. This would increase the amount of time the facility would have to abort a mission if necessary. The modified scenario lowered the maximum speed of the aircraft engine which then made the rate from waypoint to waypoint slower, increasing the amount of access time. A future modified scenario could be to change the communication systems of the aircrafts and their respective facilities.

8. Future Work

This research project started a foundation for the future capabilities of linking modeling platforms to physics-based modeling platforms. From this integration attempt, a user can upload a CSV file input from Cameo to the python script. Additionally, users can change the CSV file from Cameo to updated values and upload that to the python script. Then, by running the script, a user can view the view an automated analysis in STK and make informed decisions on how to edit the SysML Cameo model based on those results. The STK analysis currently shows graphs with the STK application and creates a CSV file that prints points from those respective graphs.

For future capabilities automating the export process out of Cameo would be a good addition to this research. This project focused more on Python integration within STK than it did within Cameo. It would be interesting and a fundamental addition to see if Cameo or SysML data in general has the capability to be automated via a coding script. In addition to this capability, fully automating the bidirectionality from Cameo to STK and STK to Cameo is one of the end goals of this research. This would lessen or eliminate communication issues that may occur from translating design data across various stakeholders.

Lastly, an extended addition to this research, once bidirectional automation is achieved, would be implementing a publisher/subscriber (pub/sub) ability for this integration. A pub/sub is used for streaming analytics and data integration pipelines to ingest and distribute data. It can be considered the “middleware” for service integration [18]. This implementation could create a notification communication that would deliver data, in this case a fictitious system design, whenever it has been updated. This pub/sub capability could also be implemented on MITRE’s software, which could create better access for MITRE and their stakeholders.

Bibliography

[1] Henderson, K, Salado, A. Value and benefits of model-based systems engineering (MBSE): Evidence from the literature. *Systems Engineering*. 2021; 24: 51– 66. https://doi.org/10.1002/sys.21566

[2] *R&D Centers*. (n.d.). MITRE. from https://www.mitre.org/our-impact/rd-centers

[3] *System and SE Definitions - Systems Engineering Definition*. (n.d.). International Council on Systems Engineering. https://www.incose.org/about-systems-engineering/system-and-se-definition

[4] *Weapon System Requirements: Detailed Systems Engineering Prior to Product Development Positions Programs for Success*. (2016, November 17). Government Accountability Office. Retrieved October 26, 2022, from https://www.gao.gov/products/gao-17-77

[5] Guide_to_the_Systems_Engineering_Body_of_Knowledge_Part_8.pdf (sebokwiki.org)

[6] *A Capability Maturity Assessment Framework for Creating High Value Digital Engineering Opportunities*. (2022, January 13). Content Delivery Network (CDN). Retrieved October 26, 2022, from https://cpb-us-w2.wpmucdn.com/wp.wpi.edu/dist/b/380/files/2022/01/A_Capability_Maturity_Assessment_Framework_for_Creating_High_Value_Digital_Engineering_Opportunities.pdf

[7] Cloutier, R. (n.d.). *Model-Based Systems Engineering Adoption Trends 2009-2018*. SEBoK. Retrieved October 26, 2022, from https://www.sebokwiki.org/wiki/Model-Based_Systems_Engineering_Adoption_Trends_2009-2018

[8] *SysML FAQ: What is SysML? What is the Systems Modeling Language?* (n.d.). SysML.org. Retrieved October 26, 2022, from https://sysml.org/sysml-faq/what-is-sysml.html

[9] *Cameo Enterprise Architecture - CATIA*. (n.d.). Dassault Systèmes. Retrieved October 26, 2022, from https://www.3ds.com/products-services/catia/products/no-magic/cameo-enterprise-architecture/

[10] *4.2 Introduction to Physics Simulation – Big Data E-Book*. (n.d.). Sites at Penn State. Retrieved October 26, 2022, from https://sites.psu.edu/bigdataebook/chapter4/04-02/

[11] *Ansys STK | Digital Mission Engineering Software*. (n.d.). Ansys. Retrieved October 28, 2022, from https://www.agi.com/products/stk

[12] Klie, H. (2021, May 3). *A Tale of Two Approaches: Physics-Based vs. Data-Driven Models*. JPT. Retrieved October 28, 2022, from https://jpt.spe.org/a-tale-of-two-approaches-physics-based-vs-data-driven-models

[13] (n.d.). U.S. Department of Defense. Retrieved October 28, 2022, from https://www.defense.gov/

[14] *UNDERSTANDING COMMAND AND CONTROL*. (n.d.). DTIC. Retrieved October 28, 2022, from https://apps.dtic.mil/sti/pdfs/ADA457162.pdf

[15] *E-3 Sentry (AWACS) > Air Force > Fact Sheet Display*. (n.d.). AF.mil. Retrieved October 28, 2022, from https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104504/e-3-sentry-awacs/

[16] *surface-to-air missile | military weapon | Britannica*. (n.d.). Encyclopedia Britannica. Retrieved October 28, 2022, from https://www.britannica.com/technology/surface-to-air-missile

[17] *Python | Pandas DataFrame*. (2019, January 10). GeeksforGeeks. Retrieved November 1, 2022, from https://www.geeksforgeeks.org/python-pandas-dataframe/

[18] *What is Pub/Sub? | Cloud Pub/Sub Documentation*. (n.d.). Google Cloud. Retrieved November 1, 2022, from https://cloud.google.com/pubsub/docs/overview

[19] (Zimmerman, P., (INCOSE MBSE Workshop, January 25, 2014). Model-Based Systems Engineering (MBSE) in Government: Leveraging the 'M' for DoD Acquisition [PowerPoint slides])

[20] (https://media.defense.gov/2021/Sep/29/2002864356/-1/-1/0/190228-F-EV310-9145.JPG)

Appendix A: Current Python Code for STK & Cameo Integration

```
from itertools import chain
from os import times
from agi.stk12.stkengine import STKEngine
from agi.stk12.stkdesktop import STKDesktop
from agi.stk12.stkobjects import *
from agi.stk12.stkutil import *

import os.path as path

# ######READING TEST########
# import csv
# import pandas as pd
# import numpy as np
# print('Adding csv information')
# Cameo_data =
open("C:\\Users\\mellis\\Documents\\MQP_9_25\\SimpleMQP3.csv", "r"); #Read
Excel File
# Cameo_data = list(csv.reader(Cameo_data))
# print(Cameo_data)

# cName = []
# cValue = []

# ctr = 0
#     #Getting information for the components of the model
# for col in Cameo_data:
#     cName.append(col[1])
#     cValue.append(col[6])
#     ctr = ctr +1
# cName = pd.Series(cName, name = 'Name')
# cValue = pd.Series(cValue, name = 'Cameo Value')
# df = pd.concat([cName, cValue], axis = 1)
# print(df)
# print(cValue.to_string(index=True))
# print('Trying to get just the numbers')
# print(cValue.to_string(index=False))
# print(cValue[8])

def startup_stk(simpleMQPScenario):
    import time
    startTime = time.time()


    #Set true to use engine, false to use gui
    useStkEngine = False

    ########################################################
    #Scenario Setup
```

```
    if (useStkEngine):
        #Launch STK Engine with NoGraphics mode
        print("Launching STK Engine...")
        stk = STKEngine.StartApplication(noGraphics = True)

        #Create root object
        stkRoot = stk.NewObjectRoot()

    else:
        #Launch GUI
        print("Launching STK...")
        stk = STKDesktop.StartApplication(visible = True, userControl =
True)

        #Get root object
        stkRoot = stk.Root

    #Set date format
    stkRoot.UnitPreferences.SetCurrentUnit("DateFormat", "UTCG")

    #Create new scenario
    print("Creating scenario...")
    stkRoot.NewScenario('MQP_Test')
    scenario = stkRoot.CurrentScenario

    #Set units
    stkRoot.ExecuteCommand('Window3D * Maximize')
    stkRoot.UnitPreferences.Item('LatitudeUnit').SetCurrentUnit('deg')
    stkRoot.UnitPreferences.Item('LongitudeUnit').SetCurrentUnit('deg')
    stkRoot.UnitPreferences.Item('Distance').SetCurrentUnit('km')

    #Set time period
    scenario.SetTimePeriod("1 Oct 2022 16:00:00", "+24hr")
    if (useStkEngine == False):
        #Graphics calls are not available when running STK Engine in
NoGraphics mode
        stkRoot.Rewind()

    totalTime = time.time() - startTime
    splitTime = time.time()
    print("--- Scenario creation: {a:4.3f} sec\t\tTotal time: {b:4.3f} sec
---".format(a=totalTime, b=totalTime))

    ############################################################
    #Import data from excel file
    import csv
    import pandas as pd
    import numpy as np

    print('Adding csv information')
    Cameo_data =
open("C:\\Users\\mellis\\Documents\\MQP_9_25\\SimpleMQP3.csv", "r"); #Read
Excel File
    Cameo_data = list(csv.reader(Cameo_data))
```

```
print(Cameo_data)

cName = []
cValue = []

ctr = 0
#Getting information for the components of the model
for col in Cameo_data:
    cName.append(col[1])
    cValue.append(col[6])
    ctr = ctr +1
cName = pd.Series(cName, name = 'Name')
cValue = pd.Series(cValue, name = 'Cameo Value')
df = pd.concat([cName, cValue], axis = 1)
print(df)
print(cValue.to_string(index=True))
print('Trying to get just the numbers')
print(cValue.to_string(index=False))
print(cValue[8])


############################################################
#Create Scenario Objects


#Creating 5 Targets Test
print("Creating 5 Targets...")
#Use NewOnCentralBody to specify explicitly the central body
areaTarget = stkRoot.CurrentScenario.Children.New(2, 'A')
#Define a boundary from list on latitude and longitude
stkRoot.BeginUpdate()
areaTarget.AreaType = 1 #ePattern
patterns = areaTarget.AreaTypeData
patterns.Add(45.00, -110.50)
patterns.Add(44.00, -109.50)
stkRoot.EndUpdate()
areaTarget.AutoCentroid = True

areaTarget2 = stkRoot.CurrentScenario.Children.New(2, 'B')
#Define a boundary from list on latitude and longitude
stkRoot.BeginUpdate()
areaTarget2.AreaType = 1 #ePattern
patterns2 = areaTarget2.AreaTypeData
patterns2.Add(43.00, -108.50)
patterns2.Add(42.00, -107.50)
stkRoot.EndUpdate()
areaTarget2.AutoCentroid = True

areaTarget3 = stkRoot.CurrentScenario.Children.New(2, 'C')
#Define a boundary from list on latitude and longitude
stkRoot.BeginUpdate()
areaTarget3.AreaType = 1 #ePattern
patterns3 = areaTarget3.AreaTypeData
patterns3.Add(41.00, -106.20)
```

```
    patterns3.Add(40.00, -105.50)
    stkRoot.EndUpdate()
    areaTarget3.AutoCentroid = True

    areaTarget4 = stkRoot.CurrentScenario.Children.New(2, 'D')
    #Define a boundary from list on latitude and longitude
    stkRoot.BeginUpdate()
    areaTarget4.AreaType = 1 #ePattern
    patterns4 = areaTarget4.AreaTypeData
    patterns4.Add(39.00, -104.50)
    patterns4.Add(38.00, -103.50)
    stkRoot.EndUpdate()
    areaTarget4.AutoCentroid = True

    areaTarget5 = stkRoot.CurrentScenario.Children.New(2, 'E')
    #Define a boundary from list on latitude and longitude
    stkRoot.BeginUpdate()
    areaTarget5.AreaType = 1 #ePattern
    patterns5 = areaTarget5.AreaTypeData
    patterns5.Add(37.00, -102.50)
    patterns5.Add(36.00, -101.50)
    stkRoot.EndUpdate()
    areaTarget5.AutoCentroid = True

    constellation =
stkRoot.CurrentScenario.Children.New(AgESTKObjectType.eConstellation,
"TargetConstellation")
    constellation.Objects.AddObject(areaTarget)
    constellation.Objects.AddObject(areaTarget2)
    constellation.Objects.AddObject(areaTarget3)
    constellation.Objects.AddObject(areaTarget4)
    constellation.Objects.AddObject(areaTarget5)

    #Creating 5 More Targets Test
    print("Creating 5 More Targets...")
    #Use NewOnCentralBody to specify explicitly the central body
    areaTarget6 = stkRoot.CurrentScenario.Children.New(2, 'F')
    #Define a boundary from list on latitude and longitude
    stkRoot.BeginUpdate()
    areaTarget6.AreaType = 1 #ePattern
    patterns6 = areaTarget6.AreaTypeData
    patterns6.Add(45, -122)
    patterns6.Add(44, -121)
    stkRoot.EndUpdate()
    areaTarget6.AutoCentroid = True

    areaTarget7 = stkRoot.CurrentScenario.Children.New(2, 'G')
    #Define a boundary from list on latitude and longitude
    stkRoot.BeginUpdate()
    areaTarget7.AreaType = 1 #ePattern
    patterns7 = areaTarget7.AreaTypeData
    patterns7.Add(43, -120)
    patterns7.Add(42, -119)
    stkRoot.EndUpdate()
```

```
    areaTarget7.AutoCentroid = True

    areaTarget8 = stkRoot.CurrentScenario.Children.New(2, 'H')
    #Define a boundary from list on latitude and longitude
    stkRoot.BeginUpdate()
    areaTarget8.AreaType = 1 #ePattern
    patterns8 = areaTarget8.AreaTypeData
    patterns8.Add(41, -118)
    patterns8.Add(40, -117)
    stkRoot.EndUpdate()
    areaTarget8.AutoCentroid = True

    areaTarget9 = stkRoot.CurrentScenario.Children.New(2, 'I')
    #Define a boundary from list on latitude and longitude
    stkRoot.BeginUpdate()
    areaTarget9.AreaType = 1 #ePattern
    patterns9 = areaTarget9.AreaTypeData
    patterns9.Add(39, -116)
    patterns9.Add(38, -115)
    stkRoot.EndUpdate()
    areaTarget9.AutoCentroid = True

    areaTarget10 = stkRoot.CurrentScenario.Children.New(2, 'J')
    #Define a boundary from list on latitude and longitude
    stkRoot.BeginUpdate()
    areaTarget10.AreaType = 1 #ePattern
    patterns10 = areaTarget10.AreaTypeData
    patterns10.Add(37, -114)
    patterns10.Add(36, -113)
    stkRoot.EndUpdate()
    areaTarget10.AutoCentroid = True

    constellation2 =
stkRoot.CurrentScenario.Children.New(AgESTKObjectType.eConstellation,
"TargetConstellation2")
    constellation2.Objects.AddObject(areaTarget6)
    constellation2.Objects.AddObject(areaTarget7)
    constellation2.Objects.AddObject(areaTarget8)
    constellation2.Objects.AddObject(areaTarget9)
    constellation2.Objects.AddObject(areaTarget10)

    ##########################################################
    #Create Aircraft
    print("Creating Aircraft")
    aircraft = scenario.Children.New(AgESTKObjectType.eAircraft,
"Aircraft1")

    #Set the Waypoints of the aircraft
    aircraft.SetRouteType(9) #ePropagatorGreatArc
    route = aircraft.Route
    route.Method = 0 #eDetermineTimeAccFromVel
    route.SetAltitudeRefType(0) #eWayPtAltRefMSL
    #Add first point
    waypoint3 = route.Waypoints.Add()
```

```
    waypoint3.Latitude = 45.30
    waypoint3.Longitude = -90
    waypoint3.Altitude = 5 #km
    waypoint3.Speed = .1 #km/sec
    #Add Target Point
    waypoint = route.Waypoints.Add()
    waypoint.Latitude = 43.00
    waypoint.Longitude = -119
    waypoint.Altitude = 5 #km
    waypoint.Speed = .1 #km/sec
    waypoint4 = route.Waypoints.Add()
    waypoint4.Latitude = 39
    waypoint4.Longitude = -100
    waypoint4.Altitude = 5 #km
    waypoint4.Speed = .1 #km/sec
    #Add return point
    waypoint2 = route.Waypoints.Add()
    waypoint2.Latitude = 45.30
    waypoint2.Longitude = -90
    waypoint2.Altitude = 100 #km
    waypoint2.Speed = .1 #km/sec
    #Adding point for out of bound scenario
    # waypointL = route.Waypoints.Add()
    # waypointL.Latitude = 42
    # waypointL.Longitude = 71
    # waypointL.Altitude = 1000 #km
    # waypointL.Speed = .20 #km/sec

    #Propagate the route
    route.Propagate()

    #Insert Antennas
    print("Creating Aircraft Antenna...")
    airAntenna = aircraft.Children.New(AgESTKObjectType.eAntenna,
"AircraftAntenna")
    #IAgAntenna antenna: Antenna object
    airAntenna.SetModel('Dipole')
    antennaModel = airAntenna.Model
    antennaModel.DesignFrequency = int(float(cValue[19])) #GHz
    antennaModel.Length = int(float(cValue[17])) #m
    antennaModel.LengthToWavelengthRatio = 45
    antennaModel.Efficiency = int(float(cValue[18])) #Percent
    #IAgAntenna antenna: Antenna object
    airAntenna.UseRefractionInAccess = True
    airAntenna.Refraction = 3 #eITU_R_P834_4
    refraction = airAntenna.RefractionModel
    refraction.Ceiling = int(float(cValue[20])) #m
    refraction.AtmosAltitude = int(float(cValue[16])) #m
    refraction.KneeBendFactor = int(float(cValue[15]))

    #Insert Transmitters
    print("Creating Aircraft Transmitter...")
    airTransmitter = aircraft.Children.New(AgESTKObjectType.eTransmitter,
"AircraftTransmitter")
```

```
    # IAgTransmitter transmitter: Transmitter object
    airTransmitter.SetModel('Complex Transmitter Model')
    txModel = airTransmitter.Model
    txModel.Frequency = int(float(cValue[11])) #GHz
    txModel.Power = int(float(cValue[14])) #dBW
    txModel.DataRate = int(float(cValue[10])) #Mb/sec
    # IAgTransmitter transmitter: Transmitter object
    txModel.EnablePolarization = True
    txModel.SetPolarizationType(3) #ePolarizationTypeLinear
    polarization = txModel.Polarization
    polarization.ReferenceAxis = int(float(cValue[12]))
#ePolarizationReferenceAxisY
    polarization.TiltAngle = int(float(cValue[13])) #deg

    ##########################################################
    #Create Second Aircraft
    print("Creating Second Aircraft")
    aircraft2 = scenario.Children.New(AgESTKObjectType.eAircraft,
"Aircraft2")

    #Set the Waypoints of the aircraft
    aircraft2.SetRouteType(9) #ePropagatorGreatArc
    route2 = aircraft2.Route
    route2.Method = 0 #eDetermineTimeAccFromVel
    route2.SetAltitudeRefType(0) #eWayPtAltRefMSL
    #Add first point
    waypoint6 = route2.Waypoints.Add()
    waypoint6.Latitude = 39
    waypoint6.Longitude = -90
    waypoint6.Altitude = 5 #km
    waypoint6.Speed = .1 #km/sec

    #Add Target Point
    waypoint5= route2.Waypoints.Add()
    waypoint5.Latitude = 43
    waypoint5.Longitude = -108
    waypoint5.Altitude = 5 #km
    waypoint5.Speed = .1 #km/sec

    waypoint7 = route2.Waypoints.Add()
    waypoint7.Latitude = 39
    waypoint7.Longitude = -100
    waypoint7.Altitude = 5 #km
    waypoint7.Speed = .1 #km/sec
    #Add return point
    waypoint8 = route2.Waypoints.Add()
    waypoint8.Latitude = 39
    waypoint8.Longitude = -90
    waypoint8.Altitude = 5 #km
    waypoint8.Speed = .1 #km/sec
    #Propagate the route
    route2.Propagate()

    #Insert Antennas
```

```
    print("Creating Aircraft Antenna...")
    airAntenna2 = aircraft2.Children.New(AgESTKObjectType.eAntenna,
"AirAntenna2")
    #IAgAntenna antenna: Antenna object
    airAntenna2.SetModel('Dipole')
    antennaModel2 = airAntenna2.Model
    antennaModel2.DesignFrequency = int(float(cValue[19])) #GHz
    antennaModel2.Length = int(float(cValue[17])) #m
    antennaModel2.LengthToWavelengthRatio = 45
    antennaModel2.Efficiency = int(float(cValue[18])) #Percent
    #IAgAntenna antenna: Antenna object
    airAntenna2.UseRefractionInAccess = True
    airAntenna2.Refraction = 3 #eITU_R_P834_4
    refraction2 = airAntenna2.RefractionModel
    refraction2.Ceiling = int(float(cValue[20])) #m
    refraction2.AtmosAltitude = int(float(cValue[16])) #m
    refraction2.KneeBendFactor = int(float(cValue[15]))

    #Insert Transmitters
    print("Creating Aircraft Transmitter...")
    airTransmitter2 =
aircraft2.Children.New(AgESTKObjectType.eTransmitter, "Transmitter2")
    # IAgTransmitter transmitter: Transmitter object
    airTransmitter2.SetModel('Complex Transmitter Model')
    txModel2 = airTransmitter2.Model
    txModel2.Frequency = int(float(cValue[11])) #GHz
    txModel2.Power = int(float(cValue[14])) #dBW
    txModel2.DataRate = int(float(cValue[10])) #Mb/sec
    # IAgTransmitter transmitter: Transmitter object
    txModel2.EnablePolarization = True
    txModel2.SetPolarizationType(3) #ePolarizationTypeLinear
    polarization2 = txModel2.Polarization
    polarization2.ReferenceAxis = int(float(cValue[12]))
#ePolarizationReferenceAxisY
    polarization2.TiltAngle = int(float(cValue[13])) #deg

    ##########################################################
    print("Creating Third Aircraft")
    aircraft3 = scenario.Children.New(AgESTKObjectType.eAircraft,
"Aircraft3")

    #Set the Waypoints of the aircraft
    aircraft3.SetRouteType(9) #ePropagatorGreatArc
    route3 = aircraft3.Route
    route3.Method = 0 #eDetermineTimeAccFromVel
    route3.SetAltitudeRefType(0) #eWayPtAltRefMSL
    #Add first point
    waypoint9 = route3.Waypoints.Add()
    waypoint9.Latitude = 34
    waypoint9.Longitude = -90
    waypoint9.Altitude = 5 #km
    waypoint9.Speed = .1 #km/sec

    #Add Target Point
```

```
    waypoint10 = route3.Waypoints.Add()
    waypoint10.Latitude = 39
    waypoint10.Longitude = -104
    waypoint10.Altitude = 5 #km
    waypoint10.Speed = .2 #km/sec

    waypoint11 = route3.Waypoints.Add()
    waypoint11.Latitude = 39
    waypoint11.Longitude = -100
    waypoint11.Altitude = 5 #km
    waypoint11.Speed = .1 #km/sec
    #Add return point
    waypoint12 = route3.Waypoints.Add()
    waypoint12.Latitude = 34
    waypoint12.Longitude = -90
    waypoint12.Altitude = 5 #km
    waypoint12.Speed = .1 #km/sec
    #Propagate the route
    route3.Propagate()

    #Insert Antennas
    print("Creating Aircraft Antenna...")
    airAntenna3 = aircraft3.Children.New(AgESTKObjectType.eAntenna,
"Antenna3")
    #IAgAntenna antenna: Antenna object
    airAntenna3.SetModel('Dipole')
    antennaModel3 = airAntenna3.Model
    antennaModel3.DesignFrequency = int(float(cValue[19])) #GHz
    antennaModel3.Length = int(float(cValue[17])) #m
    antennaModel3.LengthToWavelengthRatio = 45
    antennaModel3.Efficiency = int(float(cValue[18])) #Percent
    #IAgAntenna antenna: Antenna object
    airAntenna3.UseRefractionInAccess = True
    airAntenna3.Refraction = 3 #eITU_R_P834_4
    refraction3 = airAntenna3.RefractionModel
    refraction3.Ceiling = int(float(cValue[20])) #m
    refraction3.AtmosAltitude = int(float(cValue[16])) #m
    refraction3.KneeBendFactor = int(float(cValue[15]))

    #Insert Transmitters
    print("Creating Aircraft Transmitter...")
    airTransmitter3 =
aircraft3.Children.New(AgESTKObjectType.eTransmitter, "Transmitter3")
    # IAgTransmitter transmitter: Transmitter object
    airTransmitter3.SetModel('Complex Transmitter Model')
    txModel3 = airTransmitter3.Model
    txModel3.Frequency = int(float(cValue[11])) #GHz
    txModel3.Power = int(float(cValue[14])) #dBW
    txModel3.DataRate = int(float(cValue[10])) #Mb/sec
    # IAgTransmitter transmitter: Transmitter object
    txModel3.EnablePolarization = True
    txModel3.SetPolarizationType(3) #ePolarizationTypeLinear
    polarization3 = txModel3.Polarization
```

```
    polarization3.ReferenceAxis = int(float(cValue[12]))
#ePolarizationReferenceAxisY
    polarization3.TiltAngle = int(float(cValue[13])) #deg

    ##########################################################

    #Create Facilities
    print("Creating Command Center...")
    facility = scenario.Children.New(AgESTKObjectType.eFacility,
"CommandCenter")
    facility.Position.AssignGeodetic(42.30, -71, 0) #LAt,Long,Alt
    #Set altitude to a distance above the ground
    facility.HeightAboveGround = .1 #km

    #Add Facility Antenna
    print("Creating Command Center Antenna...")
    FAntenna = facility.Children.New(AgESTKObjectType.eAntenna,
"FacilityAntenna")
    # IAgAntenna antenna: Antenna object
    FAntenna.SetModel('Dipole')
    antennaModel = FAntenna.Model
    antennaModel.DesignFrequency = 30 # GHz
    antennaModel.Length = 1 #m
    antennaModel.LengthToWavelengthRatio = 45
    antennaModel.Efficiency = 85 #Percent
    # IAgAntenna antenna: Antenna object
    FAntenna.UseRefractionInAccess = True
    FAntenna.Refraction = 3  #eITU_R_P834_4
    refraction = FAntenna.RefractionModel
    refraction.Ceiling = 1000 #m
    refraction.AtmosAltitude = 10000 #m
    refraction.KneeBendFactor = 0.1

    #Add FAcility Transmitter
    print("Creating Command Center Transmitter...")
    FTransmitter = facility.Children.New(AgESTKObjectType.eTransmitter,
"FacilityTransmitter")

    # IAgTransmitter transmitter: Transmitter object
    FTransmitter.SetModel("Complex Transmitter Model")
    txModel2 = FTransmitter.Model
    antennaControl = txModel2.AntennaControl
    antennaControl.SetEmbeddedModel('Isotropic')
    antennaControl.EmbeddedModel.Efficiency = 85 #Percent
    # IAgTransmitter transmitter: Transmitter object
    FTransmitter.SetModel("Complex Transmitter Model")
    txModel2 = FTransmitter.Model
    txModel2.EnablePolarization = True
    txModel2.SetPolarizationType(3)  #ePolarizationTypeLinear
    polarization = txModel2.Polarization
    polarization.ReferenceAxis = 1   #ePolarizationReferenceAxisY
    polarization.TiltAngle = 15  #deg

    ##########################################################
```

```
    print("Creating North Facility...")
    facility2 = scenario.Children.New(AgESTKObjectType.eFacility,
"NorthFacility")
    facility2.Position.AssignGeodetic(45.30, -90, 0) #LAt,Long,Alt
    #Set altitude to a distance above the ground
    facility2.HeightAboveGround = .1 #km

    #Add Facility Antenna
    print("Creating North Facility Antenna...")
    FAntenna2 = facility2.Children.New(AgESTKObjectType.eAntenna,
"FacilityAntenna2")
    # IAgAntenna antenna: Antenna object
    FAntenna2.SetModel('Dipole')
    antennaModel2 = FAntenna2.Model
    antennaModel2.DesignFrequency = 30 # GHz
    antennaModel2.Length = 1 #m
    antennaModel2.LengthToWavelengthRatio = 45
    antennaModel2.Efficiency = 85 #Percent
    # IAgAntenna antenna: Antenna object
    FAntenna2.UseRefractionInAccess = True
    FAntenna2.Refraction = 3  #eITU_R_P834_4
    refraction2 = FAntenna2.RefractionModel
    refraction2.Ceiling = 1000 #m
    refraction2.AtmosAltitude = 10000 #m
    refraction2.KneeBendFactor = 0.1

    #Add FAcility Transmitter
    print("Creating North Facility Transmitter...")
    FTransmitter2 = facility2.Children.New(AgESTKObjectType.eTransmitter,
"FacilityTransmitter2")

    # IAgTransmitter transmitter: Transmitter object
    FTransmitter2.SetModel("Complex Transmitter Model")
    txModel3 = FTransmitter2.Model
    antennaControl2 = txModel3.AntennaControl
    antennaControl2.SetEmbeddedModel('Isotropic')
    antennaControl2.EmbeddedModel.Efficiency = 85 #Percent
    # IAgTransmitter transmitter: Transmitter object
    FTransmitter2.SetModel("Complex Transmitter Model")
    txModel3 = FTransmitter2.Model
    txModel3.EnablePolarization = True
    txModel3.SetPolarizationType(3)  #ePolarizationTypeLinear
    polarization2 = txModel3.Polarization
    polarization2.ReferenceAxis = 1   #ePolarizationReferenceAxisY
    polarization2.TiltAngle = 15  #deg

    ###########################################################

    print("Creating Middile Facility...")
    facilityMid = scenario.Children.New(AgESTKObjectType.eFacility,
"MiddileFacility")
    facilityMid.Position.AssignGeodetic(39, -90, 0) #LAt,Long,Alt
    #Set altitude to a distance above the ground
```

```
    facilityMid.HeightAboveGround = .1 #km

    #Add Facility Antenna
    print("Creating Middile Facility Antenna...")
    FAntennaMid = facilityMid.Children.New(AgESTKObjectType.eAntenna,
"FacilityAntenna3")
    # IAgAntenna antenna: Antenna object
    FAntennaMid.SetModel('Dipole')
    antennaModelMid = FAntennaMid.Model
    antennaModelMid.DesignFrequency = 30 # GHz
    antennaModelMid.Length = 1 #m
    antennaModelMid.LengthToWavelengthRatio = 45
    antennaModelMid.Efficiency = 85 #Percent
    # IAgAntenna antenna: Antenna object
    FAntennaMid.UseRefractionInAccess = True
    FAntennaMid.Refraction = 3  #eITU_R_P834_4
    refractionMid = FAntennaMid.RefractionModel
    refractionMid.Ceiling = 1000 #m
    refractionMid.AtmosAltitude = 10000 #m
    refractionMid.KneeBendFactor = 0.1

    #Add FAcility Transmitter
    print("Creating Middile Facility Transmitter...")
    FTransmitterMid =
facilityMid.Children.New(AgESTKObjectType.eTransmitter,
"FacilityTransmitter3")

    # IAgTransmitter transmitter: Transmitter object
    FTransmitterMid.SetModel("Complex Transmitter Model")
    txModelMid = FTransmitterMid.Model
    antennaControlMid = txModelMid.AntennaControl
    antennaControlMid.SetEmbeddedModel('Isotropic')
    antennaControlMid.EmbeddedModel.Efficiency = 85 #Percent
    # IAgTransmitter transmitter: Transmitter object
    FTransmitterMid.SetModel("Complex Transmitter Model")
    txModelMid = FTransmitterMid.Model
    txModelMid.EnablePolarization = True
    txModelMid.SetPolarizationType(3)  #ePolarizationTypeLinear
    polarizationMid = txModelMid.Polarization
    polarizationMid.ReferenceAxis = 1   #ePolarizationReferenceAxisY
    polarizationMid.TiltAngle = 15  #deg

    ##########################################################

    print("Creating South Facility...")
    facilityS = scenario.Children.New(AgESTKObjectType.eFacility,
"SouthFacility")
    facilityS.Position.AssignGeodetic(34, -90, 0) #LAt,Long,Alt
    #Set altitude to a distance above the ground
    facilityS.HeightAboveGround = .1 #km

    #Add Facility Antenna
    print("Creating South Facility Antenna...")
```

```
    FAntennaS = facilityS.Children.New(AgESTKObjectType.eAntenna,
"FacilityAntenna4")
    # IAgAntenna antenna: Antenna object
    FAntennaS.SetModel('Dipole')
    antennaModelS = FAntennaS.Model
    antennaModelS.DesignFrequency = 20 # GHz
    antennaModelS.Length = 1 #m
    antennaModelS.LengthToWavelengthRatio = 45
    antennaModelS.Efficiency = 85 #Percent
    # IAgAntenna antenna: Antenna object
    FAntennaS.UseRefractionInAccess = True
    FAntennaS.Refraction = 3  #eITU_R_P834_4
    refractionS = FAntennaS.RefractionModel
    refractionS.Ceiling = 1000 #m
    refractionS.AtmosAltitude = 10000 #m
    refractionS.KneeBendFactor = 0.1

    #Add FAcility Transmitter
    print("Creating South Facility Transmitter...")
    FTransmittersS = facilityS.Children.New(AgESTKObjectType.eTransmitter,
"FacilityTransmitter4")

    # IAgTransmitter transmitter: Transmitter object
    FTransmittersS.SetModel("Complex Transmitter Model")
    txModelS = FTransmittersS.Model
    antennaControlS = txModelS.AntennaControl
    antennaControlS.SetEmbeddedModel('Isotropic')
    antennaControlS.EmbeddedModel.Efficiency = 85 #Percent
    # IAgTransmitter transmitter: Transmitter object
    FTransmittersS.SetModel("Complex Transmitter Model")
    txModelS = FTransmittersS.Model
    txModelS.EnablePolarization = True
    txModelS.SetPolarizationType(3)  #ePolarizationTypeLinear
    polarizationS = txModelS.Polarization
    polarizationS.ReferenceAxis = 1   #ePolarizationReferenceAxisY
    polarizationS.TiltAngle = 15  #deg


    # #Compute access between first aircrat and facility
    # print("\nComputing aircraft and facility access...")
    # access2 = aircraft.GetAccessToObject(facility)
    # access2.ComputeAccess()

    # #Get the Access AER Data Provider
    # accessDP = access2.DataProviders.Item('Access Data')
    # # accessDP2 = accessDP.QueryInterface(stkobjects.IAgDataPrvInterval)
    # results = accessDP.Exec(scenario.StartTime, scenario.StopTime)
    # accessStartTimes = results.DataSets.GetDataSetByName('Start
Time').GetValues()
    # accessStopTimes = results.DataSets.GetDataSetByName('Stop
Time').GetValues()
    # print("The Access Start and End Times are: ")
    # print(accessStartTimes,accessStopTimes)
```

```
    # aircraftADP = aircraft.DataProviders.Item('Distance') #Distance Data
Provider
    # # aircraftADP2 =
aircraftADP.QueryInterface(stkobjects.IAgDataPrvInterval)
    # aircraftADPElements = ["Time to finish", "Average Speed","Dist to
finish", "Speed" ]
    # aircraftTimeVar = aircraftADP.ExecElements(accessStartTimes,
accessStopTimes, 60.0, aircraftADPElements)
    # timeToFinish = aircraftTimeVar.DataSets.GetDataSetByName("Time to
finish").GetValues()
    # averageSpeed = aircraftTimeVar.DataSets.GetDataSetByName("Average
Speed").GetValues()
    # distFinish = aircraftTimeVar.DataSets.GetDataSetByName("Dist to
finish").GetValues()
    # aircraftSpeed =
aircraftTimeVar.DataSets.GetDataSetByName("Speed").GetValues()

    # print("The first aircraft time to finish is: ")
    # print(timeToFinish)
    # print("The first aircraft average speed over time is: ")
    # print(averageSpeed)
    # print("The first aircraft distance to finish is: ")
    # print(distFinish)
    # print("The first aircraft aircraft speed over time is: ")
    # print(aircraftSpeed)

    #Compute facility antenna to aircraft antenna
    print("\nComputing access between the North Facility Antenna and the
First Aircraft transmitter...")
    access3 = FAntenna2.GetAccessToObject(airTransmitter)
    access3.ComputeAccess
    #Access Data Providers
    accessDP3 = access3.DataProviders.Item('Access Data')
    results3 = accessDP3.Exec(scenario.StartTime, scenario.StopTime)
    accessDP3Start = results3.DataSets.GetDataSetByName('Start
Time').GetValues()
    accessDP3Stop = results3.DataSets.GetDataSetByName('Stop
Time').GetValues()
    accessDP3Duration =
results3.DataSets.GetDataSetByName('Duration').GetValues()
    print("The access start and stop times are: ")
    print(accessDP3Start, accessDP3Stop)
    print("The duration of access is: ")
    print(accessDP3Duration)

    #Compute facility antenna to second aircraft antenna
    print("\nComputing access between the Middile Facility Antenna and the
Second Aircraft transmitter...")
    access4 = FAntennaMid.GetAccessToObject(airTransmitter2)
    access4.ComputeAccess
    #Access Data Providers
    accessDP4 = access4.DataProviders.Item('Access Data')
    results4 = accessDP4.Exec(scenario.StartTime, scenario.StopTime)
```

```
    accessDP4Start = results4.DataSets.GetDataSetByName('Start
Time').GetValues()
    accessDP4Stop = results4.DataSets.GetDataSetByName('Stop
Time').GetValues()
    accessDP4Duration =
results4.DataSets.GetDataSetByName('Duration').GetValues()
    print("The access start and stop times are: ")
    print(accessDP4Start, accessDP4Stop)
    print("The duration of access is: ")
    print(accessDP4Duration)

    #Compute facility antenna to second aircraft antenna
    print("\nComputing access between the South Facility Center antenna
and the Third Aircraft transmitter...")
    access5 = FAntennaS.GetAccessToObject(airTransmitter3)
    access5.ComputeAccess
    #Access Data Providers
    accessDP5 = access5.DataProviders.Item('Access Data')
    results5 = accessDP5.Exec(scenario.StartTime, scenario.StopTime)
    accessDP5Start = results5.DataSets.GetDataSetByName('Start
Time').GetValues()
    accessDP5Stop = results5.DataSets.GetDataSetByName('Stop
Time').GetValues()
    accessDP5Duration =
results5.DataSets.GetDataSetByName('Duration').GetValues()
    print("The access start and stop times are: ")
    print(accessDP5Start, accessDP5Stop)
    print("The duration of access is: ")
    print(accessDP5Duration)



    #Create Missile
    print("Creating Missiles...")
    #IAgScenario scenario: scenario object
    missile = scenario.Children.New(AgESTKObjectType.eMissile, "Missile1")
    missile.SetTrajectoryType(10) #PropagtorBallistic
    trajectory = missile.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory.Launch.Lat = 40 #Around Pennsylvania
    trajectory.Launch.Lon = -80 #Around Pennsylvania
    #Is there anyway to set taqrget without putting lat and lon?
    trajectory.ImpactLocation.Impact.Lat = 44
    trajectory.ImpactLocation.Impact.Lon = -109.50
    trajectory.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
    trajectory.Propagate()


    #Create Missile
```

```
    #IAgScenario scenario: scenario object
    missile2 = scenario.Children.New(AgESTKObjectType.eMissile,
"Missile2")
    missile2.SetTrajectoryType(10) #PropagtorBallistic
    trajectory2 = missile2.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory2.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory2.Launch.Lat = 41 #Around Pennsylvania
    trajectory2.Launch.Lon = -77 #Around Pennsylvania
    trajectory2.ImpactLocation.Impact.Lat = 42
    trajectory2.ImpactLocation.Impact.Lon = -107.50
    trajectory2.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory2.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
    trajectory2.Propagate()

    #Create Missile
    #IAgScenario scenario: scenario object
    missile3 = scenario.Children.New(AgESTKObjectType.eMissile,
"Missile3")
    missile3.SetTrajectoryType(10) #PropagtorBallistic
    trajectory3 = missile3.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory3.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory3.Launch.Lat = 40.50 #Around Pennsylvania
    trajectory3.Launch.Lon = -78 #Around Pennsylvania
    trajectory3.ImpactLocation.Impact.Lat = 40
    trajectory3.ImpactLocation.Impact.Lon = -105.50
    trajectory3.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory3.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
    trajectory3.Propagate()

    #Create Missile
    #IAgScenario scenario: scenario object
    missile4 = scenario.Children.New(AgESTKObjectType.eMissile,
"Missile4")
    missile4.SetTrajectoryType(10) #PropagtorBallistic
    trajectory4 = missile4.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory4.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory4.Launch.Lat = 41 #Around Pennsylvania
    trajectory4.Launch.Lon = -79 #Around Pennsylvania
    trajectory4.ImpactLocation.Impact.Lat = 38
    trajectory4.ImpactLocation.Impact.Lon = -103.50
    trajectory4.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory4.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
```

```
    trajectory4.Propagate()

    #Create Missile
    #IAgScenario scenario: scenario object
    missile5 = scenario.Children.New(AgESTKObjectType.eMissile,
"Missile5")
    missile5.SetTrajectoryType(10) #PropagtorBallistic
    trajectory5 = missile5.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory5.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory5.Launch.Lat = 39 #Around Pennsylvania
    trajectory5.Launch.Lon = -78 #Around Pennsylvania
    trajectory5.ImpactLocation.Impact.Lat = 36
    trajectory5.ImpactLocation.Impact.Lon = -101.50
    trajectory5.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory5.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
    trajectory5.Propagate()


    # missileSys =
stkRoot.CurrentScenario.Children.New(AgESTKObjectType.eChain,
"MissileChain")
    # chain.Objects.AddObject(missile)
    # chain.Objects.AddObject(missile2)
    # chain.Objects.AddObject(missile3)
    # chain.Objects.AddObject(missile4)
    # chain.Objects.AddObject(missile5)


    #Create Missile
    print("Creating More Missiles...")
    #IAgScenario scenario: scenario object
    missile6 = scenario.Children.New(AgESTKObjectType.eMissile,
"Missile6")
    missile6.SetTrajectoryType(10) #PropagtorBallistic
    trajectory6 = missile6.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory6.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory6.Launch.Lat = 41 #Around Pennsylvania
    trajectory6.Launch.Lon = -80.50 #Around Pennsylvania
    trajectory6.ImpactLocation.Impact.Lat = 44
    trajectory6.ImpactLocation.Impact.Lon = -121
    trajectory6.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory6.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
    trajectory6.Propagate()

    #Create Missile
    #IAgScenario scenario: scenario object
```

```
    missile7 = scenario.Children.New(AgESTKObjectType.eMissile,
"Missile7")
    missile7.SetTrajectoryType(10) #PropagtorBallistic
    trajectory7 = missile7.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory7.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory7.Launch.Lat = 40 #Around Pennsylvania
    trajectory7.Launch.Lon = -79 #Around Pennsylvania
    trajectory7.ImpactLocation.Impact.Lat = 42
    trajectory7.ImpactLocation.Impact.Lon = -119
    trajectory7.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory7.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
    trajectory7.Propagate()

    #Create Missile
    #IAgScenario scenario: scenario object
    missile8 = scenario.Children.New(AgESTKObjectType.eMissile,
"Missile8")
    missile8.SetTrajectoryType(10) #PropagtorBallistic
    trajectory8 = missile8.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory8.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory8.Launch.Lat = 39 #Around Pennsylvania
    trajectory8.Launch.Lon = -81 #Around Pennsylvania
    trajectory8.ImpactLocation.Impact.Lat = 40
    trajectory8.ImpactLocation.Impact.Lon = -117
    trajectory8.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory8.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
    trajectory8.Propagate()

    #Create Missile
    #IAgScenario scenario: scenario object
    missile9 = scenario.Children.New(AgESTKObjectType.eMissile,
"Missile9")
    missile9.SetTrajectoryType(10) #PropagtorBallistic
    trajectory9 = missile9.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory9.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory9.Launch.Lat = 40 #Around Pennsylvania
    trajectory9.Launch.Lon = -80 #Around Pennsylvania
    trajectory9.ImpactLocation.Impact.Lat = 38
    trajectory9.ImpactLocation.Impact.Lon = -115
    trajectory9.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory9.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
    trajectory9.Propagate()
```

```
    #Create Missile
    #IAgScenario scenario: scenario object
    missile10 = scenario.Children.New(AgESTKObjectType.eMissile,
"Missile10")
    missile10.SetTrajectoryType(10) #PropagtorBallistic
    trajectory10 = missile7.Trajectory
    stkRoot.UnitPreferences.SetCurrentUnit('DateFormat', 'EpSec')
    trajectory10.EphemerisInterval.SetExplicitInterval(0, 0)  # stop time
later computed based on propagation
    trajectory10.Launch.Lat = 42 #Around Pennsylvania
    trajectory10.Launch.Lon = -79 #Around Pennsylvania
    trajectory10.ImpactLocation.Impact.Lat = 36
    trajectory10.ImpactLocation.Impact.Lon = -113
    trajectory10.ImpactLocation.SetLaunchControlType(0)
#eLaunchControlFixedApogeeAlt
    trajectory10.ImpactLocation.LaunchControl.ApogeeAlt =
int(float(cValue[7])) #km
    trajectory10.Propagate()

    # missileSys2 =
stkRoot.CurrentScenario.Children.New(AgESTKObjectType.eChain,
"MissileChain2")
    # chain.Objects.AddObject(missile6)
    # chain.Objects.AddObject(missile7)
    # chain.Objects.AddObject(missile8)
    # chain.Objects.AddObject(missile9)
    # chain.Objects.AddObject(missile10)

    #Scenario 2 will be changing the missile altitude and seeing how it
affects overall time

    # #Access of ONE missile to its Target
    missileADP = missile.DataProviders.Item('Ground Range')
    missileADP2 = missileADP.Group.Item('Fixed')
    missileElements = ['Time', 'Ground Range', 'Alt']
    missileTimeVar = missileADP2.ExecElements(scenario.StartTime,
scenario.StopTime, 60.0, missileElements)
    missileTime =
missileTimeVar.DataSets.GetDataSetByName('Time').GetValues()
    missileGroundRange = missileTimeVar.DataSets.GetDataSetByName('Ground
Range').GetValues()
    missileAlt =
missileTimeVar.DataSets.GetDataSetByName('Alt').GetValues()

    print("The Missile Time Is: ")
    print(missileTime)
    print("The Missile Ground Range Over Time Is: ")
    print(missileGroundRange)
    print("The Missile Altitude Over Time is: ")
    print(missileAlt)


    #Writing the scenario analysis data to a CSV to be viewed
```

```
    from csv import writer
    from csv import reader
    import pathlib
    import pandas as pd
    import matplotlib.pyplot as plt
    import numpy as np

    #Looking to get the values of the Aircraft

    #Looking to get the values of the Missile Range vs Missile Altitude
    x = pd.Series(missileGroundRange, name = 'Ground Range (km)' )
    y = pd.Series(missileAlt, name = 'Altitude (km)')
    z = pd.Series(missileTime, name = 'Time (sec)')
    #Concatenate to coloums
    df = pd.concat([x, y,z], axis = 1)
    print(df)
    csv_data = df.to_csv("C:\\Users\\mellis\\Documents\\MQP
Resources\\MQP_Missile_Scenario1.csv")
    print(csv_data)
    #Plot the coordinates DOES NOT WORK, plots taken from STK
    #df.plot(y = 'yValues', figsize=(10,6), title ='Missile Altitude vs
Ground Range', xlabel ='Range (km)', ylabel = 'Altitude')
    fpath = "C:\\Users\\mellis\\Documents\\STK 12\\simpleMQScenario"
    startup_stk(fpath)
```