NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EXECUTABLE MBSE APPROACH WITH ILLUSTRATION OF A SATELLITE ENGAGEMENT MISSION DESIGN

by

Diego C. Rangel

June 2021

Thesis Advisor:	Oleg A. Yakimenko
Co-Advisor:	Saulius Pavalkis, Dassault Systemes
Second Reader:	Fotis A. Papoulias

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2021	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE EXECUTABLE MBSE APPROACH WITH ILLUSTRATION OF A SATELLITE ENGAGEMENT MISSION DESIGN	**5. FUNDING NUMBERS**
6. AUTHOR(S) Diego C. Rangel	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Model-based systems engineering (MBSE) is becoming the industry standard for systems engineering activities. To avoid design flaws and reduce rework and cost, the descriptive models developed using the modern MBSE tools need to be integrated with other engineering discipline models. The co-simulation approach envisioned for current MBSE tools facilitates the use of external solvers to solve mathematical expressions within the model. Indeed, integrating complex simulations to couple descriptive and physics-based models is a challenging task requiring quite a few adjustments to both models to produce an executable MBSE model. This thesis aims to enhance the use of one of the most advanced MBSE tools—Cameo Systems Modeler (CSM)—to be able to execute high-fidelity models of combat systems running in the Simulink development environment. Such an executable model should greatly improve and enhance feasibility of analysis of any combat mission during early system design phases. As an example, this thesis models a co-orbital engagement (COE) of two satellites and walks through all steps of the CSM-Simulink integration process. A shared workspace of MATLAB serves as a critical enabler for dealing with the data transfer. The thesis provides an example of how the developed integrated model can be used to analyze the COE mission and explore an effect of reshaping the design space via varying a set of mission requirements.

14. SUBJECT TERMS MBSE, Cameo Systems Modeler, Simulink, executable models, mission analysis	**15. NUMBER OF PAGES** 95
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

EXECUTABLE MBSE APPROACH WITH ILLUSTRATION OF A SATELLITE ENGAGEMENT MISSION DESIGN

Diego C. Rangel
Lieutenant, Brazilian Navy
BNS, Brazilian Naval Academy, 2010
NE, University of Sao Paulo, 2015

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2021

Approved by: Oleg A. Yakimenko
Advisor

Saulius Pavalkis
Co-Advisor

Fotis A. Papoulias
Second Reader

Ronald E. Giachetti
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Model-based systems engineering (MBSE) is becoming the industry standard for systems engineering activities. To avoid design flaws and reduce rework and cost, the descriptive models developed using the modern MBSE tools need to be integrated with other engineering discipline models. The co-simulation approach envisioned for current MBSE tools facilitates the use of external solvers to solve mathematical expressions within the model. Indeed, integrating complex simulations to couple descriptive and physics-based models is a challenging task requiring quite a few adjustments to both models to produce an executable MBSE model. This thesis aims to enhance the use of one of the most advanced MBSE tools—Cameo Systems Modeler (CSM)—to be able to execute high-fidelity models of combat systems running in the Simulink development environment. Such an executable model should greatly improve and enhance feasibility of analysis of any combat mission during early system design phases. As an example, this thesis models a co-orbital engagement (COE) of two satellites and walks through all steps of the CSM-Simulink integration process. A shared workspace of MATLAB serves as a critical enabler for dealing with the data transfer. The thesis provides an example of how the developed integrated model can be used to analyze the COE mission and explore an effect of reshaping the design space via varying a set of mission requirements.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ASAT	anti-satellite weapon
BDD	block definition diagram
COE	co-orbital engagement
CSM	Cameo Systems Modeler
CST	Cameo Simulation Toolkit
CW	Clohessy-Wiltshire
ESA	European Space Agency
GEO	geosynchronous Earth orbit
GN&C	guidance, navigation, and control
GUI	graphical user interface
LEO	Low Earth orbit
MBSE	model-based systems engineering
MEO	medium Earth orbit
MET	mission engineering thread
MoE	measure of effectiveness
MoP	measure of performance
OV	overview
RIC	radial, in-track, cross-track
RPO	rendezvous and proximity operations
SoI	system of interest
UI	user interface
UML	unified modeling language
USAF	United States Air Force

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Model-based systems engineering (MBSE) is a "subset of digital engineering" that "supports the systems engineering activities of requirements, architecture, design, verification, and validation [1]." In MBSE, the systems engineers of a complex system create a descriptive model that serves as the single source of truth to avoid inconsistencies. The MBSE approach is rapidly replacing the traditional document-centric approach and becoming the industry standard to execute systems engineering activities. Despite this fact, to be truly effective, it is necessary to connect these descriptive models to other engineering disciplines' physics-based models. This integration mitigates design flaws and helps the requirements verification, reducing rework and cost. According to [1], this integration of descriptive and physics-based models to produce high-fidelity executable system models remains challenging for digital engineering.

In their conception, military systems are usually complex enough to require this sophisticated approach. The Department of Defense Digital Engineering Strategy acknowledges the necessity to "incorporate technological innovations into an integrated, digital, model-based approach" to transform its engineering practices and improve the military acquisition process to ensure technological superiority [2]. The strategy's first goal is to formalize the development, integration, and use of models to support analysis and decisions [2]. This usage of models aims to facilitate decisions made during the early design stages without the necessity of mock-ups or physical testing. So, executable model analysis shortens the feedback loop and allows a quicker and cheaper way to evaluate countless more design variations than the design-build-test traditional approach [3].

There are currently two approaches for deriving executable models from static architecture views: co-simulation and model transformation [4]. Using the co-simulation approach, many commercial MBSE tools such as Cameo Systems Modeler (CSM), IBM Rhapsody, and Innoslate have implemented features that allow integration of the descriptive and other design disciplines' models for simulation purposes. Several frameworks in the literature on transforming static architecture models into executable models describe model transformation [4]. One can consider model transformation more

complicated than co-simulation since it is less intuitive. According to [5], model transformation requires a "good knowledge of metamodels, and the link between the elements defined in the metamodel and concrete syntax." Besides, the co-simulation level achieved in commercial tools seems to facilitate the design of executable models without requiring much effort from these software users, which is why this thesis focuses on this approach. Another option is workflow automation, currently available in tools such as Phoenix Integration Model Center and Dassault Systemes Process Composer. These tools do not co-simulate but automatically transforms the descriptive model.

Specifically, this thesis is devoted to enhancing CSM, a traditional commercially available MBSE software, through its integration with multidomain simulations of system's dynamics developed in MATLAB/Simulink development environment. To illustrate the proposed executable model approach, this thesis adopts the co-orbital engagement (COE) between satellites as the mission to be modeled to demonstrate the integration process and show how valuable an executable model can be even during early development phases. To this end, the following paragraph introduces the COE problem.

Satellites contribute decisively to nations' conduct of military operations, and warfare tactics rely more and more on space asset's information. Space warfare is already a reality, and there is a myriad of anti-satellite (ASAT) weapons and techniques. From 2015 to 2020, co-orbital ASAT weapons have been a topic in the headlines. These weapons put an interceptor into orbit that can realize maneuvers to alter its orbit, allowing the interceptor to get closer to its target [6]. Due to satellites' complexity, their design requires more technological tools to ensure that stakeholder needs and user expectations are addressed in the best way possible, especially those used for military purposes. In the aerospace industry, the end-product and all the support operations necessary to place a satellite in orbit are so expensive that, in most cases, prototyping and testing of more than one design variant aspect are budget prohibitive. The emergence of new threats adds new challenges that require evaluating tactics and other operational aspects of a system, never tested before, in its early development stages. So, it is vital to ensure that the modeled design can satisfy mission requirements before the design team expends time and resources to prototype the system.

Based on publicly available information about COE, a vignette for the COE mission is presented and used for modeling a descriptive model of this mission in CSM. After that, this descriptive model is integrated with a legacy model that simulates two satellites' behavior in COE [7]. This thesis revealed that the best approach is to use the physics-based models as black boxes nested in the CSM descriptive model to provide more flexibility for modelers and the use of a shared workspace in MATLAB, allowing the data exchange between the CSM and Simulink. In this approach, opaque actions in activity diagrams are the interface for inputs and outputs between the Simulink model and the CSM model. The integration is considered successful since, after the process explained in the thesis, the descriptive model rules the physics-based model and presents the simulation outputs for mission analysts. The user can simulate COE missions through a graphical user interface (GUI) or an instance table in the developed executable model.

Despite the success in the integration, there are still some limitations related to the blocks' value properties in the descriptive model. First, CSM does not support value properties written in the matrix form, which is considered a major limitation for coding in MATLAB. The updating of value properties in CSM and variables in the shared workspace also requires additional effort since unified modeling language (UML) commands are necessary. Another CSM GUI limitation is the lack of integration capability to display graphs and images generated in the MATLAB environment. The thesis presents ways to overcome these issues and other important technical details for those interested in integrating descriptive and physics-based models into executable models.

This thesis recommends further studies scaling the developed model to include more complex simulations in the same descriptive model and explore their interaction. This interaction can occur so that the data and outputs generated by one physics-based model serve as inputs to the other physics-based model. This new interaction may require modification in designing the data flow in the developed model to keep the descriptive model updated. Additionally, this model would evolve to include a high-fidelity model developed in another Cameo Simulation Toolkit (CST) supported scripting language.

References

[1] R. Giachetti, "Digital engineering," *The Guide to the Systems Engineering Body of Knowledge* (SEBoK), v. 2.3 R.J. Accessed Feb. 12, 2021. [Online]. Available: https://www.sebokwiki.org/wiki/Digital_Engineering

[2] Office of the Deputy Assistant Secretary of Defense for Systems Engineering, "Digital engineering strategy," Department of Defense, Washington, DC, USA, 2018. [Online]. Available: https://fas.org/man/eprint/digeng-2018.pdf

[3] B. Stone, "There is no spoon: U.S. Air Force digital acquisition strategy (Summary)," You Tube, Feb. 07, 2021. [Online]. Available: https://www.youtube.com/watch?v=dEcPlqImjWc&feature=youtu.be

[4] M. Amissah, "A framework for executable systems modeling," Ph.D. dissertation, Dept. of Engineering Management, Old Dominion University, Norfolk, VA, USA, 2018. [Online]. Available: https://digitalcommons.odu.edu/emse_etds/31/

[5] B. Chabibi, A. Anwar, and M. Nassar, "Towards a model integration from SysML to MATLAB/Simulink," *J. Softw.*, vol. 13, no. 12, pp. 630–645, Dec. 2018. [Online]. doi: 10.17706/jsw.13.12.630-645.

[6] B. Weeden and K. Pfrang, "Russian co-orbital anti-satellite testing," Secure World Foundation, August 2020. [Online]. Available: https://swfound.org/media/207051/swf_russian_co-orbital-asat_aug2020.pdf

[7] E. A. Hanlon, "Design strategies and tactics to defeat co-orbital anti-satellite capabilities," M.S. thesis, Dept. of Syst. Eng., Naval Postgraduate School, Monterey, CA, USA2018. [Online]. Available: https://apps.dtic.mil/sti/pdfs/AD1059897.pdf

ACKNOWLEDGMENTS

To my wife, Lorena Galm. Thank you for all the love and affection dedicated throughout this journey. Your support was essential at all stages, and without you, I would never be able to make this dream come true.

To my parents, Clodomiro and Alecia, my first educators. Thank you for teaching me that any goal can be achieved with effort and dedication, and for being so supportive my whole life.

I would also like to thank my co-advisor, Dr. Saulius Pavalkis, who taught me how to use Cameo Systems Modeler and helped me address and resolve a variety of integration issues.

To my thesis advisor, Dr. Oleg Yakimenko, I do not have words to express how grateful I am for all the trust placed in my work. Thank you for your patience, support, and guidance. I am genuinely honored to have the opportunity to work with you.

THIS PAGE INTENTIONALLY LEFT BLANK

I. BACKGROUND AND PROBLEM FORMULATION

This chapter explains the importance of the executable model-based systems engineering (MBSE) approach to assess behaviors of complex systems such as satellites correctly. After that, the chapter introduces the co-orbital engagement (COE) problem in the context of space warfare. It explains why this subject raises concerns for the operational maintenance of the current space infrastructure and how the topic impacts on future spacecraft design. The chapter ends by presenting an executable MBSE approach for COE missions and summarizes the thesis structure.

A. EXECUTABLE MODEL-BASED SYSTEM ENGINEERING APPROACH

According to [1], MBSE "is a subset of digital engineering" that "supports the systems engineering activities of requirements, architecture, design, verification, and validation." The MBSE approach facilitates the maintenance, synchronization, and assessment of the information generated about a complex system and formalizes systems engineering processes using a model. The models developed using an MBSE tool are scalable and reusable, capable of evolving to support the system's operation throughout its life cycle. Moreover, the use of a descriptive model as the single source of truth avoids the inconsistencies created when using the traditional document-centric approach.

To truly avoid design flaws, however, it is necessary to connect these descriptive models to other engineering disciplines' physics-based models, which can ultimately reduce rework and cost. This integration of descriptive and physics-based models to produce high-fidelity executable system models remains a challenge for digital engineering [1]. Executable system's models accelerate the learning curve, enable higher-quality models, and facilitate trade-offs aimed to optimize the system as a whole [2]. On the other hand, disconnected models lead to misinformation within the design team, non-coherent or unrealistic design analysis, interface problems, and lack of integration between subsystems.

For complex man-made systems, executable models also help the identification of a system's emergent behaviors facilitating decisions made during the early design stages

without the necessity of expensive mock-ups or physical testing. In other words, executable model analysis shortens the feedback loop and allows a quicker and cheaper way to evaluate countless more design variations than is possible with the design-build-test traditional approach [3]. According to [4], this substitution of real-world activities by digital ones constitutes the art of digital acquisition.

Specifically, according to [5], "the state of the art with regards to derivations of executable models from static architectures views is the two approaches of Model Transformation and Co-Simulation." Current commercial MBSE tools such as the Cameo Systems Modeler (CSM), IBM Rhapsody, and Innoslate have features that allow integration of the descriptive and other design disciplines' models for simulation purposes. These tools "provide out of the box support for co-simulation using scripting languages" [5] like Python and MATLAB/Simulink. The co-simulation approach provides a simpler way to create executable models and allows various domain experts to operate and test aspects of the system concurrently. This way of designing systems was unthinkable using the traditional document-centric approach or only static representations of them.

Any model transformations use a process that generates "an equivalent model in a target language based on a specified mapping between the source and the target languages [5]." According to [6], model transformation requires a "good knowledge of metamodels, and the link between the elements defined in the metamodel and concrete syntax" makes model transformation a more complex task than the co-simulation approach. Several proposed approaches and frameworks in the relevant literature transform static architecture models into executable models [5]. Most of them aim to create executable models having SysML as the source language.

Workflow automation is a more straightforward option for model transformation. This alternative is currently available in tools such as Phoenix Integration Model Center and Dassault Systemes Process Composer. These tools do not co-simulate but automatically transforms the descriptive model. This integration through automated workflows adds another software in the toolchain to produce an executable model. On the other hand, it does not require deep knowledge of metamodels.

Both ways to develop executable models present shortcomings and particular challenges. However, the co-simulation level achieved with commercial tools seems to facilitate the development of effective executable models without requiring much effort from these software users. This is the main reason why the co-simulation approach using two well-known industry tools is the one chosen to develop an executable model in this thesis.

This thesis is devoted to enhancing the use of one of the most advanced MBSE tools, CSM, and executing high-fidelity models running in a development environment most domain engineers are using these days – MathWorks's MATLAB/Simulink. As an illustration, this thesis adopts the Simulink co-orbital engagement model and shows how to effectively use it in the conceptual design of the COE missions. To this end, the following two sections introduce the COE problem.

B. SPACE WARFARE

Satellites have become indispensable for several modern human activities such as navigation, communications, and weather forecasts. Besides all these implementations of satellite technology taken for granted in modern society, satellites also represent a strategic and tactical advantage for the nations that have them, since they offer a global perspective of the operational picture during conflicts. In military use, satellites can determine the enemy's troop movements, missile warning systems, and weapons' guidance, and can be used in targeting enemies' assets. These are just a few examples of exclusively military capabilities improved by satellites' data.

For these reasons, space technology contributes decisively to nations' conduct of military operations, and warfare tactics rely more and more on the usage of space assets' information. This dependency creates the necessity for these countries to ensure their space infrastructure is always active and updated. At the same time, this reliance on data obtained from satellites turns these spacecraft assets into a potential target for adversaries.

Due to their inhospitable operational environment and systems' characteristics, spacecraft have many vulnerabilities that can be exploited by adversaries. Even nascent space powers can exploit these weaknesses and use them as a leverage tool against

established powers [7]. Besides that, repairing a spacecraft in orbit is not economically viable yet, which means that even a simple malfunction in a support system can make a spacecraft useless. Replacement is not an easy option either. The production and the deployment of spacecraft involve various variables that make their replacement unfeasible on a tactical time scale.

The protection of this critical infrastructure gained increasing attention after the 2000s due to the major space powers' demonstrations of new anti-satellite (ASAT) capabilities, such as the Chinese test of a direct ascent missile in 2007 [8] or the recent concerns raised by the possible Russian test of a co-orbital weapon in 2017 [9]. These offensive capabilities are driving the formulation of new requirements and tactics for satellites, which will significantly affect the next generation of satellites' design. According to [8], artificial intelligence and improved sensors will increase the situational awareness for autonomous self-protection. Consequently, while military planners develop new tactics, spacecraft developers need to incorporate innovative design features and robustly assess their performance in the early stages of design. This is crucial, especially considering the weight and volume constraints imposed to reduce launch costs.

Yet, it is a challenge to test the suitability of new tactics never used before for a complex system such as a satellite in a brand-new warfare domain and, simultaneously, to forecast their impacts in design and operational requirements. The best approach to overcome this issue is using modeling tools capable of representing systems and their missions both structurally and logically associated with physics-based simulations. MBSE is an approach that has many benefits for the development of complex systems, especially in the aerospace sector. Nevertheless, it is necessary to promote greater integration of the descriptive models generated through the MBSE approach and other models capable of capturing the physics involved in the appropriate use of these systems to create more robust model execution.

The development of executable models can perfectly accomplish the task of integrating and testing both tactics and new designs, creating a common ground for technical and operational discussion. According to [10], recent analysis work has demonstrated the great potential of examining simultaneously operational system models

and system synthesis models. Besides, in the early system development life cycle, models are valuable since they allow a virtual analysis of solutions before engineering teams physically prototype them. These initial models can also mature to support their physical counterparts' verification, validation, operations, modernization, and logistical support.

The proper design of such an executable model involves a good understanding of both the system and its mission. So, it is crucial to identify the current threats to the system and their capabilities as a first step. The following section describes the present and near-future status of co-orbital ASAT based only on publicly available information.

C. EMERGING SPACE THREATS

There is a myriad of ASAT weapons and techniques. From 2015 to 2020, co-orbital ASAT weapons became a notable topic in the headlines. These weapons put an interceptor into orbit that can realize maneuvers to alter its orbit, thus allowing the satellite to get closer to its target [11]. Co-orbital ASATs have versatility as one of their prime benefits [12]. They can eliminate their target using the kinetic energy in a direct "kamikaze-style" impact or release a cloud of fragments that will damage the target spacecraft similarly to a multiple debris' collision. Since their modus operandi involves rendezvous and proximity operations (RPO), it is also possible for the ASAT to sabotage the target through a robotic arm capable of disassembling components or implanting small explosives; they can jam communications or use direct energy weapons. Co-orbital ASAT systems' attacks can also be more surgical, generating less debris than direct-ascent missiles [13]. Furthermore, it is challenging to identify orbit objects' hostile intentions, since their activities can be dormant long before the attack [14].

Depending on the nature of the attack, its linkage to a specific nation is not easy either [12]. Nevertheless, RPO require a high level of development in space technology, making this form of attack almost exclusively accessible to space's leading powers. According to [15], only three countries have the technological maturity for co-orbital capabilities: Russia, China, and the United States. The assessment presented by [16] confirms this assumption for the current and near-future capabilities of those three nations, using only publicly available information for that. Figure 1 compiles the counterspace

capabilities of eight nations. Russia presents significant low Earth orbit (LEO) co-orbital ASAT capabilities, while China and the United States demonstrate some. For medium and geosynchronous Earth orbit (MEO/GEO) co-orbital ASAT capabilities, the three nations raised yellow alert flags, as highlighted in Figure 1.

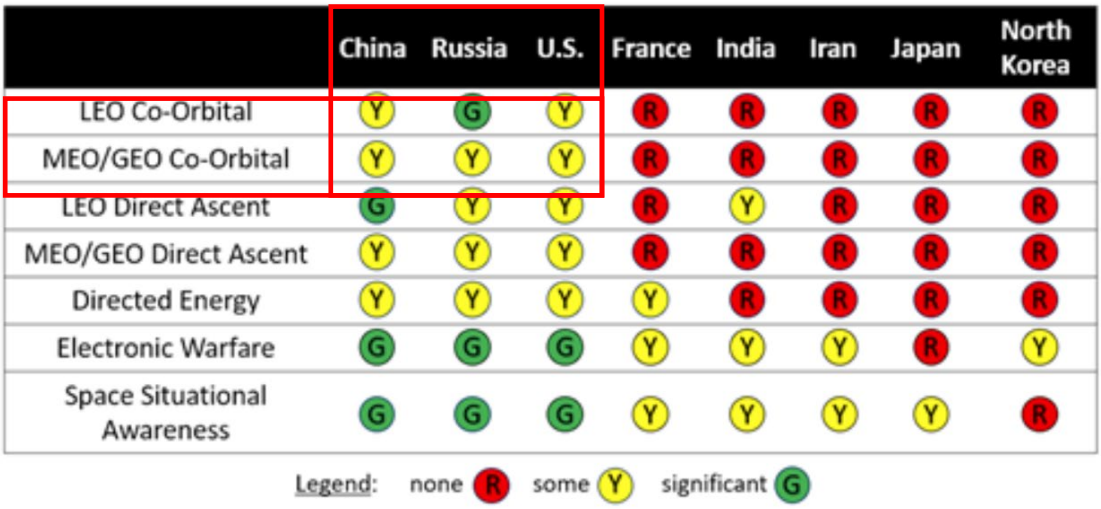

Figure 1. Overall counterspace capabilities assessment. Adapted from [16].

It is essential to highlight that RPO is a desired capability, since it improves the maintenance and inspection of space assets, making space exploration more economically viable. It is also easier for a space power to mask from other nations the development of new weapons through RPO, claiming that their tests aim on-orbit repairs [12]. In the past few years, some Russian and Chinese spacecraft have conducted unusual maneuvers in space that raised international suspicions that both nations are testing co-orbital engagement capabilities [17].

Since 2010, Russia has been conducting tests for rendezvous and close approaches in both LEO and GEO, which raised some concerns from other nations. An emblematic example is the Russian satellite "Luch" (Luch/Olymp for the U.S. Air Force). Luch has maneuvered to approach other satellites in the GEO belt since 2014, most of them communications satellites [17]. In 2018, the French Defense Minister accused the Russian

spacecraft of espionage when it made a “too close approach” to the Athena-Fidus, a dual use military telecommunications satellite that serves both French and Italian armed forces. The approach occurred as part of a Luch’s movement to get closer to Paksat-1R, a Pakistan communications satellite [17], [18]. According to [17], “Luch has parked near more than a dozen commercial communications satellites for periods ranging from a few weeks to nine months,” usually within their uplink window. Figure 2 shows Luch’s orbital history and the name of the satellites it visited. From 2014 to 2021, Luch visited 24 satellites from different countries.

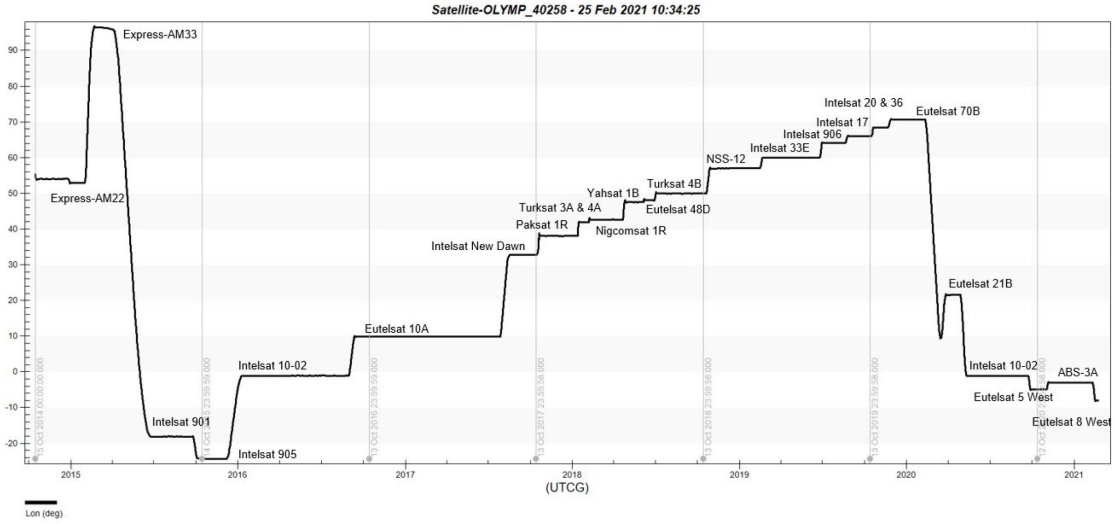

Figure 2. Compilation of Luch’s orbital history and satellites visited.
Source: [17].

In addition to Luch’s suspicious movements, the activities of other satellites designated as Cosmos are most concerning, since they provide strong evidence of non-destructive on-orbit weapons tests, and ASAT interceptors tests [9] [17]. For example, the RPO activities of Cosmos 2535 and Cosmos 2536 generated unexplained orbital debris [17]. Additionally, Cosmos 2543 deployed an object at high relative velocity of between 140 and 186 meters per second on July 2015, and, similarly, Cosmos 2321 deployed Cosmos 2523 at high-speed on October 2017 [17]. United States Space Command stated that the Russian satellites presented space-based weapons’ characteristics [17]. Table 1

summarizes the recent Russian RPO since 2014. From June 2014 to October 2020, space observers could notice a series of unusual maneuvers from Russian satellites, including the emergence of debris in the vicinity of some of them.

Table 1. Russian satellites' suspicious co-orbital tests. Adapted from [17].

Date	System(s)	Notes
Jun. 2014 – Mar. 2016	Cosmos 2499, Briz-KM R/B	Cosmos 2499 did a series of maneuvers to bring it close to, and then away from, the Briz-KM upper stage.
Apr. 2015 – Apr. 2017	Cosmos 2504, Briz-KM R/B	Cosmos 2504 maneuvers to approach the Briz-KM upper stage and may have had a slight impact before separating again.
Mar. – Apr. 2017	Cosmos 2504, FY-1C Debris	After a year dormancy, Cosmos 2504 did a close approach with a piece of Chinese space debris from the 2007 ASAT test.
Oct. 2014 – Feb. 2020	Luch, Multiple	Luch parked near several satellites over nearly five years, including the Russian Express AM-6, U.S. Intelsat 7, Intelsat 401, Intelsat 17, Intelsat 20, Intelsat 36, and French-Italian Athena-Fidus satellites.
Aug. – Oct. 2017	Cosmos 2521, Cosmos 2519, Cosmos 2523	Cosmos 2521 separated from Cosmos 2519 and performed a series of small maneuvers to do inspections before redocking with Cosmos 2519. Cosmos 2523 separated from Cosmos 2521 but did not maneuver on its own.
Mar. – Apr. 2018	Cosmos 2521, Cosmos 2519	Cosmos 2521 conducted close approaches of Cosmos 2519.
Dec. 2019 – Mar. 2019	Cosmos 2542, Cosmos 2543, USA 245	Cosmos 2542 released Cosmos 2543. Cosmos 2543 did station keeping with Cosmos 2542, then raised its orbit to come within 30 km of USA 245 and establish repeated close approaches within 150 km, likely for the purpose of surveillance. Cosmos 2542 also made close approaches to USA 245.
Jun. – Oct.2020	Cosmos 2543, Cosmos 2535, Cosmos 2536	Cosmos 2543 rendezvoused with Cosmos 2535 and released a small object at high relative velocity. In Sept., Cosmos 2536 joined in the RPO with the other two and may have docked with Cosmos 2535.

According to a U.S.-China Economic and Security Review Commission report, in 2015 [13], China has been conducting on-orbit demonstrations of rendezvous between satellites since 2010 [17]. The same commission [13] highlights that China's manned space program could justify this space technology, since it has both military and non-military applications. However, the report also states that "the secrecy surrounding the tests suggest China also is using the tests to develop co-orbital counterspace technologies" [13]. Table 2 summarizes the latest Chinese RPO demonstrations. China's recent demonstrations are consistent for the purpose of satellite servicing and inspection but are also concerning due their suitability for offensive actions against another spacecraft.

Table 2. Recent Chinese rendezvous and proximity operations. Adapted from [15].

Date	System(s)	Notes
Jun. – Aug. 2010	SJ-06F, SJ-12	SJ-12 maneuvered to rendezvous with SJ-06F. Satellites may have bumped into each other.
Jul. 2013 – May 2016	SY-7, CX-3, SJ-15	SY-7 released an additional object that it performed maneuvers with and may have a telerobotic arm. CX-3 performed optical surveillance of other in-space objects. SJ-15 demonstrated altitude and inclinations changes to approach other satellites.
Nov. 2016 – Feb. 2018	SJ-17	SJ-17 demonstrated maneuverability around the GEO belt and circumnavigated Chinasat 5A.
Jan. – Apr. 2019	TJS-3, TJS-3 AKM	TJS-3 AKM separated from the TJS-3 in the GEO belt, and both performed small maneuvers to maintain relatively close orbital slots.

Although it has no openly acknowledged co-orbital ASAT program or intent to develop such capability, the United States has the latent technological capability to develop this kind of attack in a relatively short time [19]. In fact, during the Delta 180 experiment in the 1980s, the United States conducted a successful co-orbital intercept [19]. Besides that, many RPO between satellites in both LEO and GEO demonstrate the country's knowledge in tracking and targeting objects in space [19].

Even though [16] has assessed that Japan has no co-orbital ASAT capability, that country has shown enough technological maturity to develop this capability in the near future. In April 2019, Japan deployed to Ryugu asteroid the Hayabusa-2 probe system equipped with a small carry-on impactor containing a 14-kilogram plastic explosive [20]. Hayabusa's mission is to collect asteroid samples and bring them back to Earth [21]. It is relatively easy, however, to turn this capability into a co-orbital ASAT capable of exploding a deployed satellite.

D. EMPLOYING THE EXECUTABLE MBSE APPROACH FOR COE MISSION DESIGN

Now that the COE problem has been discussed, let us return to the discussion started in Section A. Due to satellites' complexity, their design requires more technological tools to ensure that stakeholder needs and user expectations are addressed in the best way possible, especially the design of satellites used for military purposes. In the aerospace industry, the end-product and all the support operations necessary to place a satellite in orbit are so expensive that, in most cases, prototyping and testing of more than one design variant aspect are budget prohibitive. The emergence of new threats adds new challenges that require evaluating tactics and other operational aspects of the new, untested system, in its early development stages. So, it is vital to ensure that the modeled design can satisfy mission requirements before the design team expends time and resource prototyping the system.

The best way to ensure that is the adoption of a model-based approach to identify design flaws, perform trade studies, and forecast the system's emergent behavior by the design team. In fact, the Department of Defense Digital Engineering Strategy emphasizes the necessity to "incorporate technological innovations into an integrated, digital, model-based approach" to transform its engineering practices to improve the military acquisition process and ensure technological superiority [22]. In this context, the strategy's first goal is to formalize the development, integration, and use of models to support analysis and decisions [22].

Use of MBSE approach requires determining the modeling language to standardize the communication among stakeholders, the modeling tool to construct the model, and a modeling process to guide the design team. So, there are several ways to implement the MBSE approach. In this thesis, the modeling language adopted is SysML, since this unified modeling language (UML) derived language has become a standard among MBSE practitioners [23], and the chosen modeling tool is CSM. The process is a simplified version of the Mission Engineering process presented in [24]. Figure 3 conveys the steps of this process. The first two steps from which all the modeling derives are the problem statement and the mission characterization and metrics. The problem statement presents a capability gap that the system aims to satisfy. The previous two sections cover most of the information necessary to understand the capability gap created by the vulnerability to COE ASAT weapons. In the second step, from the definition and understanding of the problem, subject matter experts create a vignette and define metrics to measure system success in this vignette to explore the system's design-space. After that, the design team models both the mission and system architecture, defines requirements, and decide the appropriate simulations to use in the design of the analysis phase. This is the phase in which the executable model is designed. The last two steps address the simulation runs and the design decisions, as well as the conclusions are derived from them. The COE illustration (Figure 3) presented in this thesis covers all these phases.

Figure 3. Adopted process. Adapted from [24].

Other researchers and design teams have already reinforced the importance of executable models for satellites' operations, such as the European Space Agency (ESA) e.Deorbit mission. European Space Agency (ESA) applied this philosophy since the first

phase of that project, where contractors were requested to model physical, logical, and functional architectures using MBSE and start simulations directly from the MBSE model [25]. Spangelo et al. [26] presented capabilities to improve the CubeSat missions' design and operation using MBSE and SysML, highlighting the importance of interfacing SysML models integrated with analysis tools. Nevertheless, no current academic research discusses the same for military satellites.

E. OBJECTIVES

While there are many examples utilizing the MBSE approach in describing a system or its mission, no real attempt to incorporate a high-fidelity model into conceptual design has been documented. Thus, this thesis tackles the design of an executable model integrating a traditional MBSE tool (CSM) with a widely used external evaluator (MATLAB/Simulink) for the COE mission assessment, ensuring the necessary realism of the modeled system behavior. To address this problem, the thesis answers the following research questions:

1. Is it possible to employ high-fidelity models integrated with descriptive models to improve the assessment of systems' missions during preliminary design through executable MBSE models?

2. Can CSM be integrated with MATLAB/Simulink models for the assessment of military satellite missions?

3. What are the remaining challenges and current limitations in this kind of integration?

4. What are the benefits of such an approach?

Again, this thesis aims to demonstrate how assessing a specific satellite's mission, such as the COE between two satellites, is facilitated during conceptual design using the executable MBSE approach that employ a high-fidelity model. This thesis covers all steps necessary to design an executable model using CSM and MATLAB/Simulink, a well-known development environment in the aerospace industry. The thesis also demonstrates

the model's use through a trade study analysis that considers both system and mission aspects.

F. THESIS ORGANIZATION

To address the research questions formulated in the previous section, the remainder of this thesis is organized as follows.

Chapter II examines the COE mission's descriptive model. This chapter presents the co-orbital engagement vignette and its modeling using a MBSE approach. It highlights the SysML diagrams used to represent the co-orbital engagement mission and its operational requirements. It also provides explanations about other SysML diagrams developed to support mission analysis.

Chapter III presents important aspects of the high-fidelity Simulink model used to simulate a co-orbital engagement between two satellites. This simulator developed by [12] is used here for integration proof-of-concept purposes. Its integration with the descriptive model aims to provide system' stakeholders with an executable model for their mission analysis.

Chapter IV highlights the critical aspects of the integration of the two chosen modeling tools: CSM and MATLAB/Simulink. This chapter focuses on providing the technical details and best practice for those interested in integrating descriptive and physics-based models into executable models.

Chapter V provides an example of the use of the executable model developed, and lastly, Chapter VI presents conclusions and recommendations for those interested in pursuing the development of executable MBSE models, and further work ideas based on this thesis's conclusions.

THIS PAGE INTENTIONALLY LEFT BLANK

II. DESCRIPTIVE COE MODEL

This chapter starts with a brief description of CSM and the SysML language for readers not familiar with them. After that, it presents the developed COE evaluation vignette. In the context of the COE mission, this chapter proceeds by showing how general SysML diagrams can be applied to this specific mission derived from the evaluation vignette and other diagrams necessary to support the COE mission analysis.

A. CAMEO SYSTEMS MODELER

The Cameo Systems Modeler used in this thesis is one version of MagicDraw designed explicitly to support the MBSE approach. It provides a collaborative environment that allows design teams to visualize a system's aspects in the most standard-compliant SysML diagrams; it hides the UML-related part with customizable menus [27], [28]. The CSM plug-in Cameo Simulation Toolkit (CST) enables validation and verification of a system's dynamic aspects throughout user interaction and execution scenarios [28], [29].

CST integrates with external math engines, allowing external evaluators to evaluate opaque expressions in SysML activity and state machine diagrams [30]. MATLAB is one of the supported languages which permits the use of Simulink, the industry's well-known multi-domain simulation environment, to analyze a system's behavior using physical-based models. The integration enhances the solver capabilities already embedded in CSM and makes all plotting and animated functions from MATLAB available to facilitate data analysis and decision making. The use of MATLAB as a solver only for mathematical expressions is quite simple. On the other hand, integrating complex physics-based simulations is more challenging, requiring certain technical details to fully integrate both models within an executable one.

B. SYSML LANGUAGE AND DIAGRAMS

According to [31], to properly represent systems structure and behavior, it is necessary to use a language that shares common properties with the phenomena we are

trying to convey. Due to that, systems engineers use modeling languages to improve communication within the design team and capture different perspectives of the system.

There are a few modeling languages available. Among them, SysML is the dominant architecture modeling language, and it was developed specifically to support systems engineering activities such as specification, analysis, design, verification, and validation [23]. SysML is a general-purpose graphical modeling language derived from UML, which is commonly used in software engineering disciplines. The language is not only capable of representing system structures and behaviors, but also capturing a system's requirements and parametrics through nine different diagrams. Figure 4 shows the SysML diagram taxonomy.

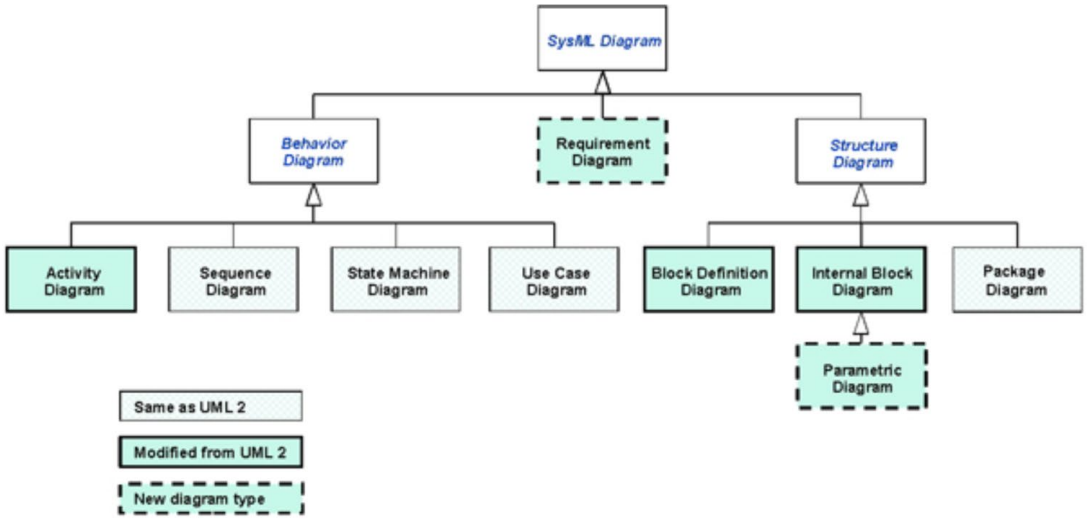

Figure 4. SysML diagram taxonomy. Source: [32].

Each diagram has its own characteristics to present different views of the system. According to [33], the diagram addresses a particular purpose, and the modeler is free to choose what should or should not be shown in this view. The absence of an element in a diagram does not mean the element does not exist in the system [23], it only means that, for that view, the design information is not necessary or was chosen to not be modeled for a particular reason. Also, there is no obligation to use all the diagrams when describing a system or a particular aspect of the system.

Furthermore, SysML provides the necessary semantics to be integrated with other engineering analysis models [32]. The great advantage of using SysML language to model a system lies in the fact that it can be integrated as the source of information for other analysis and simulation tools [32]. The following section presents the descriptive model produced for the system's mission analysis.

C. EVALUATION VIGNETTE

The vulnerability of satellites to co-orbital attacks is the system's capability gap that needs analysis. Since this is a new threat, the development and test of new tactics are complex. Only through simulations can analysts verify their assumptions and which design parameters most affect them. However, to develop a simulation for this kind of analysis, it is necessary to characterize the system's operation through a vignette and define the metrics used to assess the system.

By definition, a vignette looks at one aspect of the scenario in which the system is inserted and operates, thus providing necessary information such as events, behaviors, systems interactions, and environmental factors [24]. A vignette is usually described textually in a paragraph or two, and graphically through a context diagram [34]. A proposed vignette aims to help the design team address the issue raised in the identification of new threats for the system.

A co-orbital engagement involves, by definition, RPO, which means that one manner to avoid it is maneuvering to keep the distance between the hostile spacecraft and the target spacecraft. Space maneuvers imply fuel consumption, however, which is a critical resource associated with the satellite's lifespan in the calculations. So, the development of tactics to avoid the co-orbital engagement between satellites directly impacts the spacecraft design itself. Considering the current space scenario described in the previous chapter, one possible vignette is as follows.

During normal operations, the system of interest (SoI) detects another spacecraft's unusual presence in its orbit. SoI assumes hostile intentions and classifies the target as an ASAT, since the detected spacecraft has initiated rendezvous maneuvers. The SoI also starts to maneuver, aiming to prevent the rendezvous. Considering the criticality of fuel,

the defender aims to use any opportunity to save fuel and maximize the attacker's fuel consumption. The attacker can be equipped with a grappling arm or not. There is no interference from other spacecraft during the engagement.

Figure 5 depicts a High-Level Operational Concept (Operational View, OV) Graph of the co-orbital engagement, highlighting the interactions between the two satellites and the Earth's gravitational influence. From this vignette, the design team can derive operational requirements and measures of effectiveness at the system level that allow the design evaluation in this specific task.

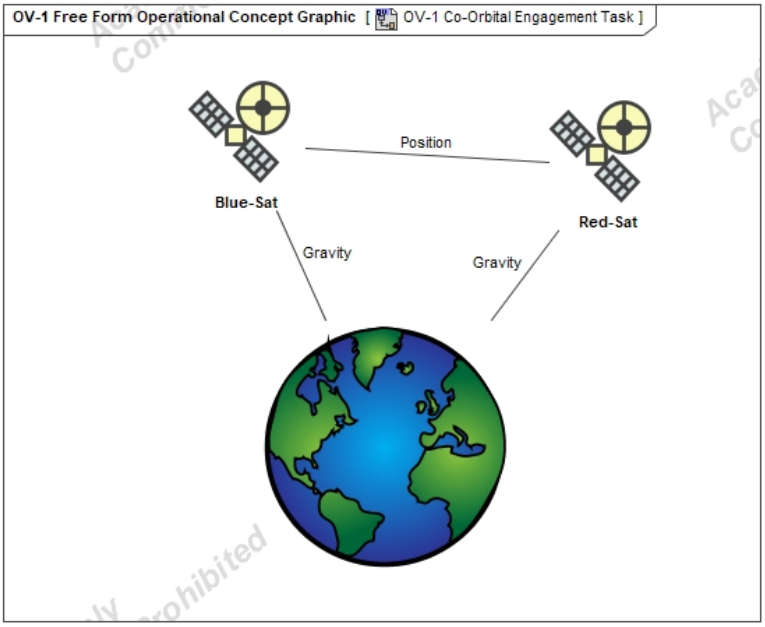

Figure 5. COE OV-1.

Metrics are quantities used to examine, determine, and track a system's performance and the value or utility it achieves in a mission [24] [35]. To track the system's suitability for the mission, this example employs only two categories of metrics: measures of effectiveness (MoE) and measures of performance (MoP). While the MoEs measure success within the overall mission, the MoPs indicate individual systems' or subsystems' performance [24]. One MoE and three MoPs constitute the metrics employed in the design of analysis. The percentage of successful evasions is the MoE. The MoPs are the minimal

distance between the satellites, the defender's fuel consumption trying to evade, and the attacker's fuel consumption trying to engage.

D. COE MISSION MODEL

The descriptive model employs SysML diagrams to describe the necessary architectures in the design of analysis phase. A top-down approach is used, starting at the mission level and breaking it down into the system and subsystem levels. It is crucial to clarify how the developed model's hierarchy applies these terms, since a system at one level can be a subsystem or a component at other levels [36]. The model's hierarchy considers both satellites as systems and their part properties as subsystems. There is no further decomposition into the components level, although there is nothing in the model that prohibits this decomposition if necessary. The following paragraphs present the products produced in this phase in their logical production flow and justify the SysML diagram used in each case. There are also explanations of other SysML and UML diagrams produced to support the run model and conclusion phases.

a) Mission Engineering Thread (MET): Missions are sets of operational tasks that need to be performed successfully for the mission's successful completion [35]. The MET defines the chain of events that the subsystems interact with to complete tasks against threats to achieve success, providing a reference for analysis and evaluation [24]. This chain of events also shows the allocation of tasks to subsystems that will be part of the SoI architecture before its definition. The SysML diagram representing the MET is the activity diagram, since this behavioral diagram conveys a dynamic vision of the system that expresses sequences of behaviors and events. Figure 6 shows the developed MET.

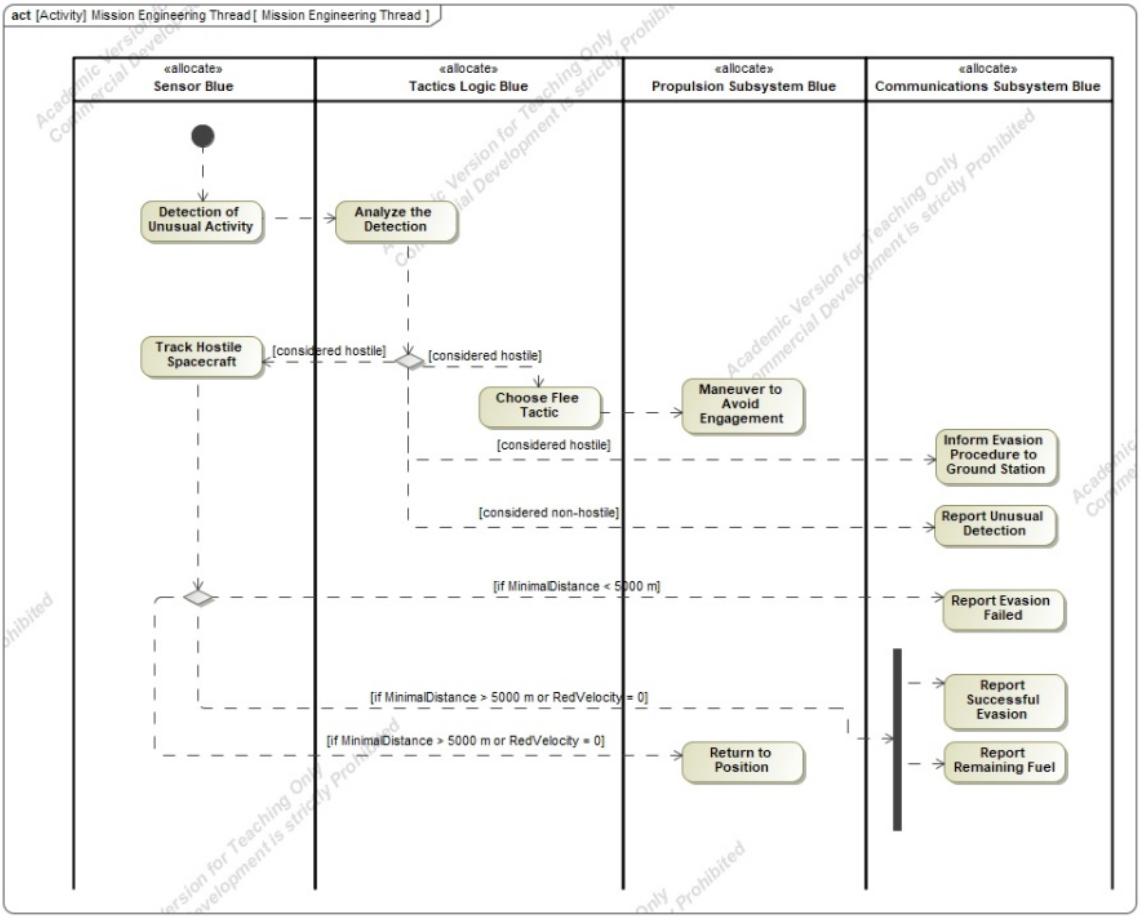

Figure 6. Mission engineering thread.

b) Mission and Systems Structural Decomposition: Block Definition Diagrams (BDD) convey the structural decomposition of the mission and both systems, and blocks only represent subsystems respecting the hierarchy just mentioned. Figure 7 illustrates the model's hierarchical levels and presents how to identify them in the descriptive model. The figure depicts associated system levels and their counterparts in the model. There is no further decomposition below the subsystem level.

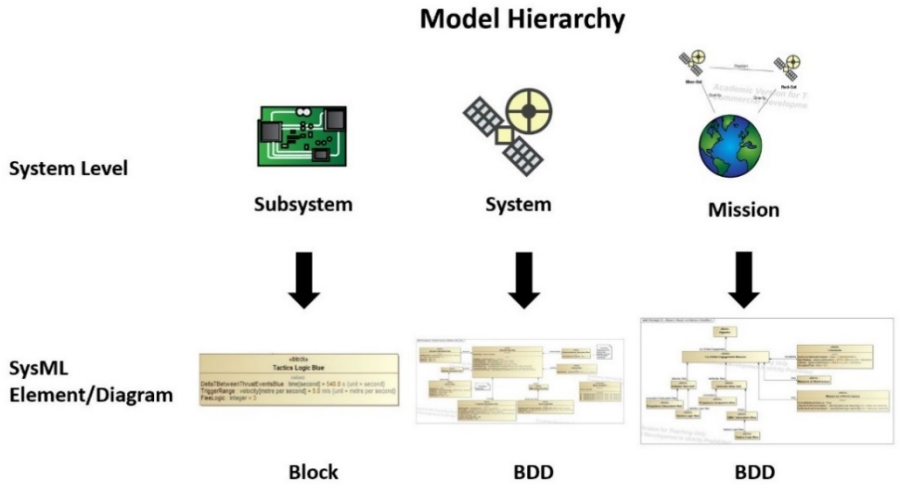

Figure 7. Model hierarchy and SysML representation of each level.

The first BBD (Figure 8) aims to show every system, entity, and metric that compose the mission. This BDD is the context of the simulation explained in the next chapter, since this diagram provides a top view that encompasses all the elements analyzed. There is one block listing all physical constants employed in the simulation, and it also highlights the most important subsystems for this mission in each spacecraft.

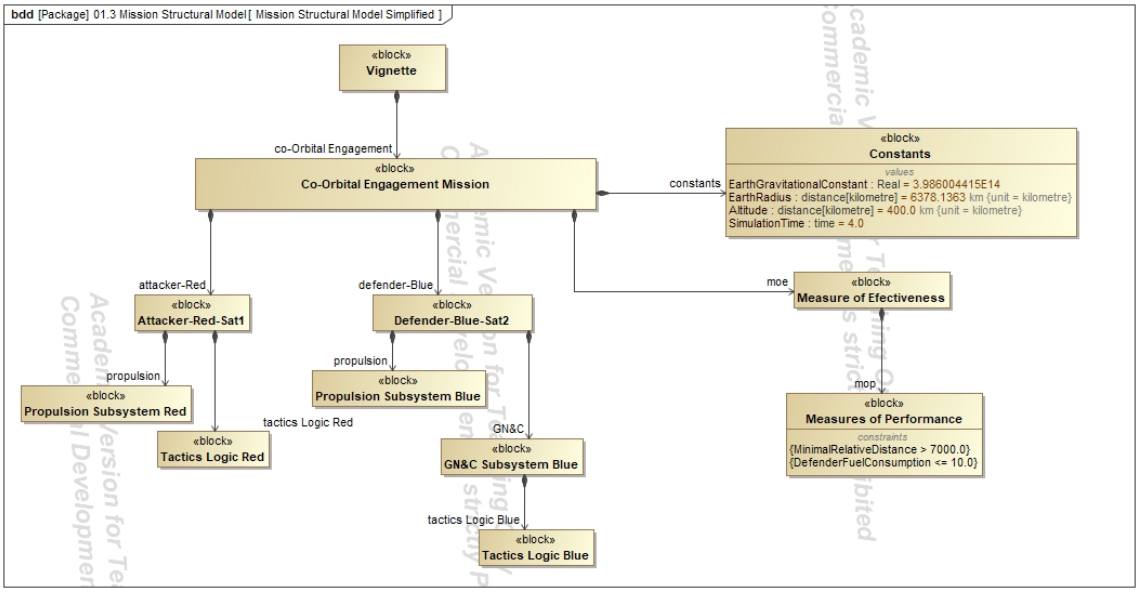

Figure 8. Co-orbital engagement mission structural decomposition.

The lower architectural level zooms in on the SoI and the ASAT structural decomposition. These BDDs expand the structure presented in the previous Mission BDD, showing generic and commonly found subsystems in satellites. Figure 9 shows the ASAT architecture.

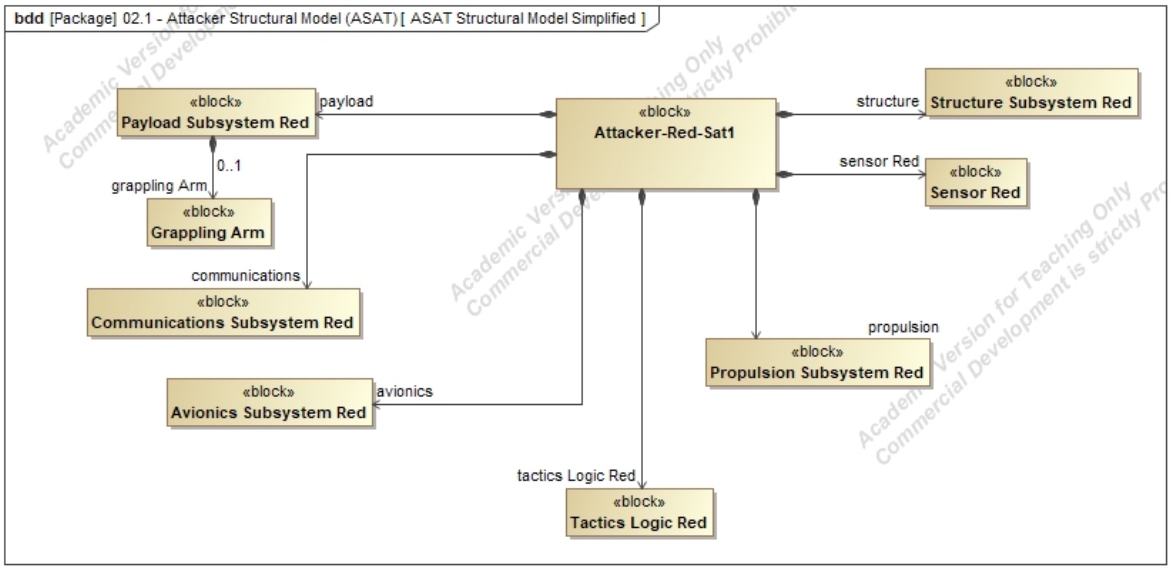

Figure 9. ASAT structural decomposition.

Figure 10 conveys the SoI structural decomposition. The propulsion and the guidance, navigation, and control (GN&C) subsystems are the only two subsystems that send inputs to the simulator explained in Chapter III. The other subsystems presented in the decomposition reflect a typical space community decomposition adopted in [37].

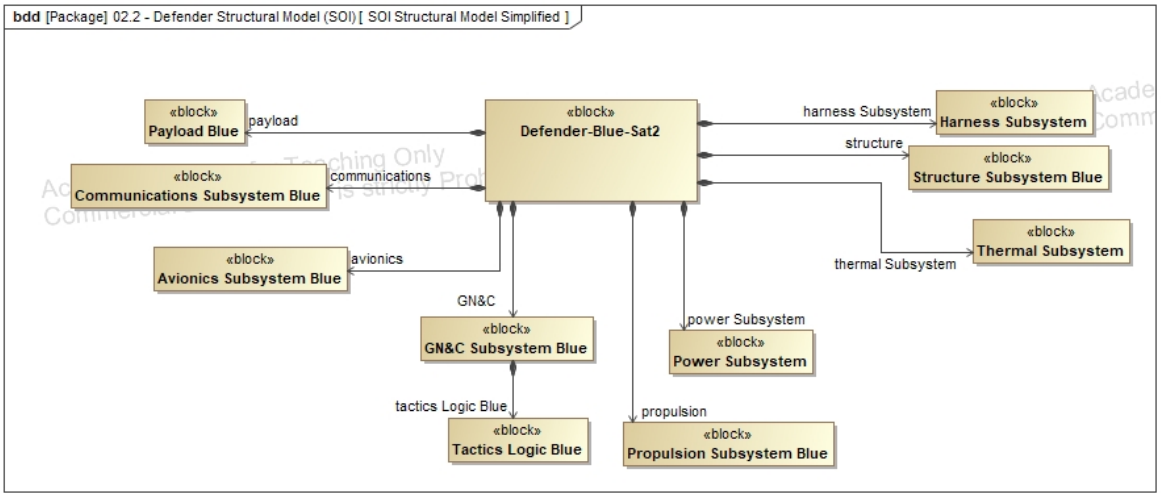

Figure 10. SoI structural decomposition.

c) Operational Requirements: Systems engineering processes usually define requirements before the structural decomposition of the system. However, since this is a mission-driven example where the material solution is already defined, the inverted order is not a problem for the process adopted in this thesis.

The requirement diagram conveys the text-based requirements and their relationship with other model elements [23]. They are useful to establish requirements traceability within the model. Figure 11 shows a simplified operational requirements diagram. These requirements reflect system-specific performance and quantify the MoPs previously defined. Usually, point values are not appropriate, and a range of values should be used instead [35]. The "objective" value is the design goal. The "threshold" value represents a superior or a lower limit not to question the utility due to the requirement aspect to which it refers to.

For simplicity, only three requirements were included in this model to illustrate the benefits of using the MBSE approach to analyze different designs. The first one is the mass requirement, which is a driver for most space missions, and the other two requirements address the system's goals related to the proposed vignette. The requirements diagram also shows that two subsystems, defined in the SoI structural decomposition, satisfy the

operational requirements: the tactics logic inside the GN&C subsystem and the propulsion subsystem.

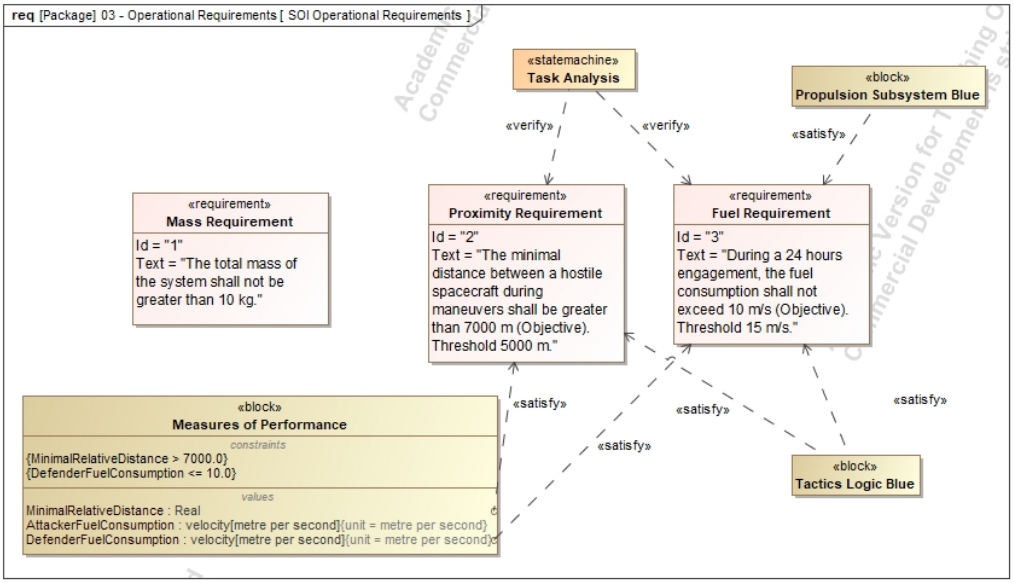

Figure 11. SoI operational requirements.

d) Requirements Verification: A benefit of the MBSE approach is that it facilitates requirements checks. A parametric diagram is a SysML diagram capable of capturing system constraints through mathematical expressions, improving the analysis by automatically checking the fulfillment of the requirements. Since there are two requirements for mission success, a parametric diagram checks both (Figure 12). It also updates the Boolean value properties inside the MOE block to register success or failure considering the simulation results for a set of mission and system parameters. Another parametric diagram in the model is responsible for the mass roll-up of the SoI. The illustration later in Chapter V, however, does not consider the mass requirement, only the requirements related to the COE mission.

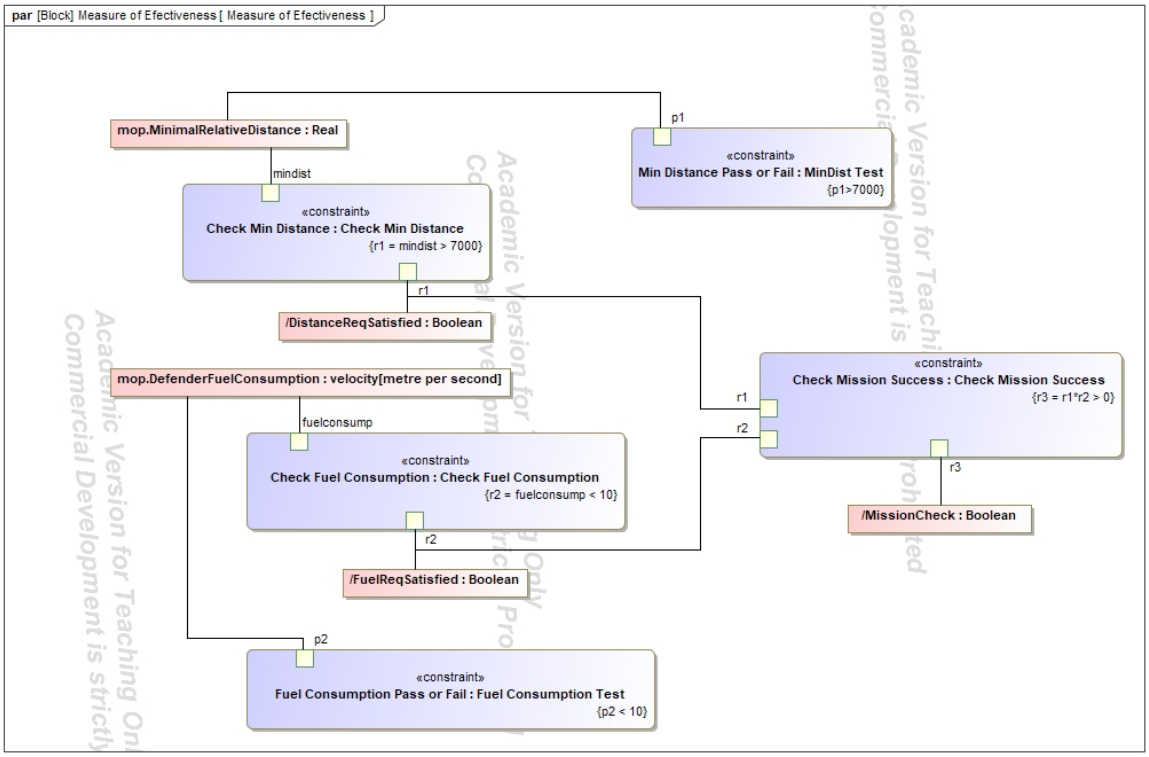

Figure 12. Parametric diagram for mission success or fail check.

e) Analysis Support Diagrams: Mission analysis employs analytical and computational means using data and models [24]. Since using an external evaluator to facilitate this analysis is the goal of this thesis, it is necessary to create the environment for this integration within the descriptive model. With that in mind, two diagrams were created to provide the user interface and its logic in the usage of analytical tools to verify system performance.

e1) State Machine Diagram: As in an activity diagram, state machine diagrams represent a dynamic view of the system which focuses on response changes to event occurrences [23]. Usually, a block owns the state machines diagram and executes this diagram within the context of a block's instance [33]. The diagram defines block behavior changes due to the transition between different states and within a state [33].

Since the high-fidelity Simulink model described in Chapter III simulates both SoI and ASAT behavior in the co-orbital engagement, in the thesis's example this behavioral diagram addresses the state's transitions associated with the commands in the graphical

user interface (GUI) that aids the mission analysis. The diagram itself is designed in a logical sequence to use MATLAB/Simulink as an external evaluator. This diagram is further explained in Chapter IV.

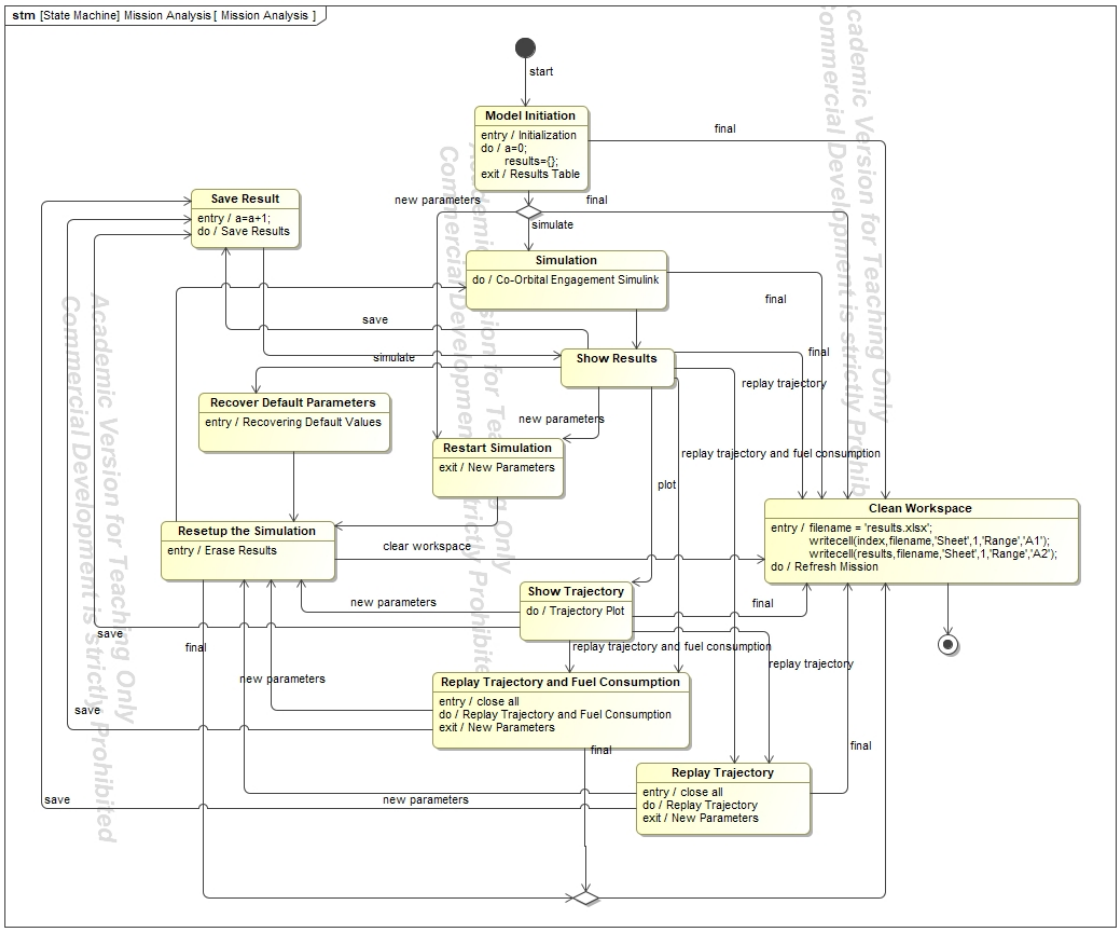

Figure 13. State machine diagram for the mission analysis.

e2) User Interface Diagram: This is an extension diagram of UML supported by CSM. A User Interface (UI) Modeling diagram allows the prototype of user interfaces. This user-friendly graphical interface allows non-technical stakeholders to understand and profit from the model in their analysis and decisions. The GUI and its association with other parts of the descriptive model are described more fully in Chapter IV.

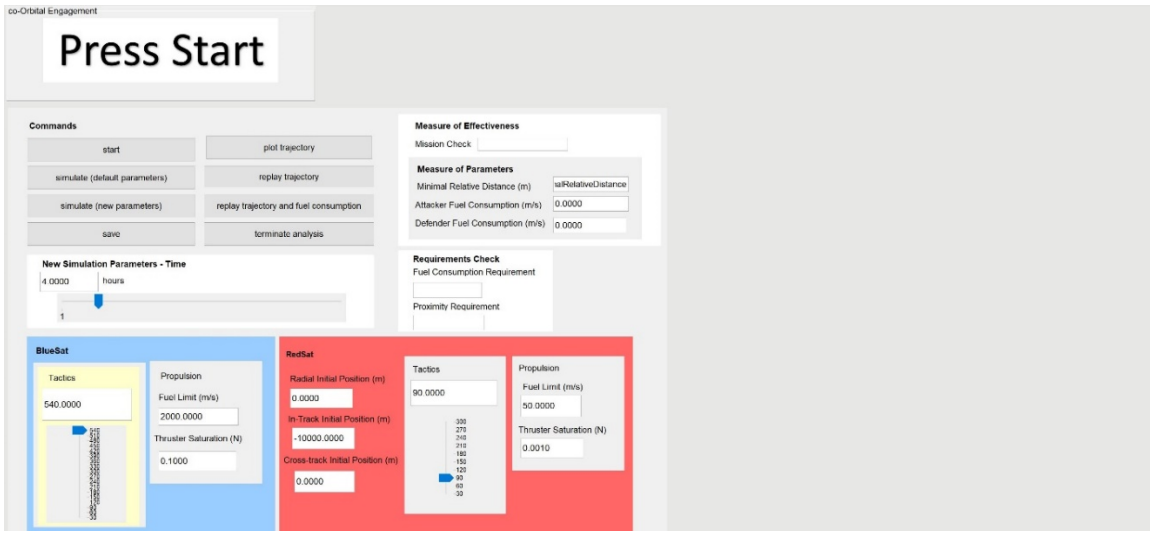

Figure 14. User Interface for co-orbital engagement assessment.

e3) Instance Table Diagram: Despite all the benefits of a user interface, CSM has a better tool for the quicker exploration of the design space. The instance table is a MagicDraw native modeling tool table that allows a user to manage various instance specifications of the model simultaneously in a spreadsheet format [38]. In this example, each row of the instance table represents an instance of the analyzed block, and its columns represent the value properties of the block chosen for display. It is possible to edit values and run simulations directly from the table. Also, CSM exports the table's data into .html, .cvs, or *.xlsx files [38], which facilitates the post-processing of the information generated in the model. Chapter IV further describes the table setting and its use.

e4) Simulation Configuration Diagram: The simulation configuration diagram conveys elements in the model related with aspects of the simulation itself. In the descriptive model, this diagram nests two simulation configuration elements and an image switcher. The simulation configuration element helps the user to customize simulation options and the image switcher is a predefined UI configuration element that helps to provide animation representing states of the system. Chapter IV discusses the role of both the simulation configuration element and image switcher in the GUI.

f) Model Overview: (Package Diagram) This diagram aims to facilitate navigation through the model (Figure 15). The better the organization, the more manageable the

control and reuse of the model. The package hierarchy is logically defined by the flow of the systems engineering activities related to the mission analysis.

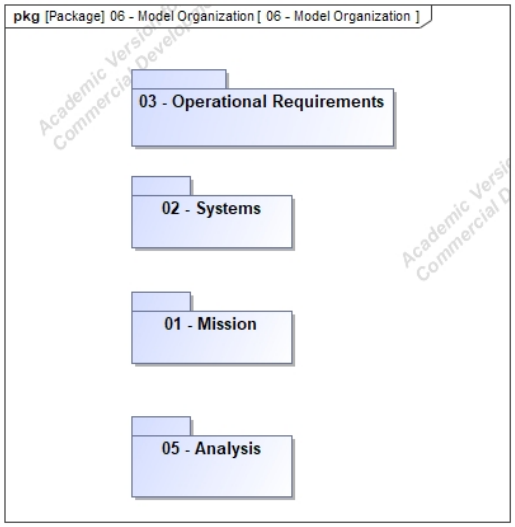

Figure 15. Model organization.

III. PHYSICS-BASED MODEL COE MODEL

This chapter describes a high-fidelity model developed in the Simulink development environment to analyze COEs between two spacecraft [12]. It starts with contextualizing the necessity of this kind of modeling and where this physics-based model fits int the descriptive model. Then, the chapter provides a brief overview of the Simulink model itself and its relationship with the SoI's and ASAT's structural parts. The final section of the chapter emphasizes a few modifications that need to be made in the Simulink model to make it run in CSM.

A. RELATIONSHIP BETWEEN THE DESCRIPTIVE AND PHYSICS-BASED MODELS

The SysML behavior diagrams, introduced in Section D of the previous chapter, do not provide a physics-based representation of the system's behavior. However, the conceptual design analysis may require this level of fidelity. This is definitely true for the space maneuvers that should be simulated exhaustively to guarantee the spacecraft design's correctness at the mission level long before the construction of any prototype.

While the descriptive model provides an overview of the entire mission, the physics-based model explained in this chapter represents the system behavior associated with the evasion tactic being tested and the maneuvers during evasion. The simulation considers GN&C parameters inside the tactics block and parameters from the propulsion subsystem that are crucial in a rendezvous maneuver. Hence, the Simulink model serves as an analytical tool to verify the fulfillment of mission requirements relative to these two subsystems' design parameters. Figure 16 highlights the tasks the model covers within the MET. The only two tasks being assessed are the evasion tactic and the maneuvers to avoid the engagement. While the evasion tactic explicitly affects the overall mission parameters and requirements satisfaction, the maneuvers characteristics affect the design of both the GN&C and the propulsion subsystem.

The physics-based model used in this study is a slightly modified version of the engagement simulator presented in [12]. Since the MBSE is scalable, the tasks allocated to

other subsystems can have their own analytical assessment tools and models, which are not within the scope of this thesis. Their integration with the descriptive model, however, follows the same procedures as those presented in Chapter IV.

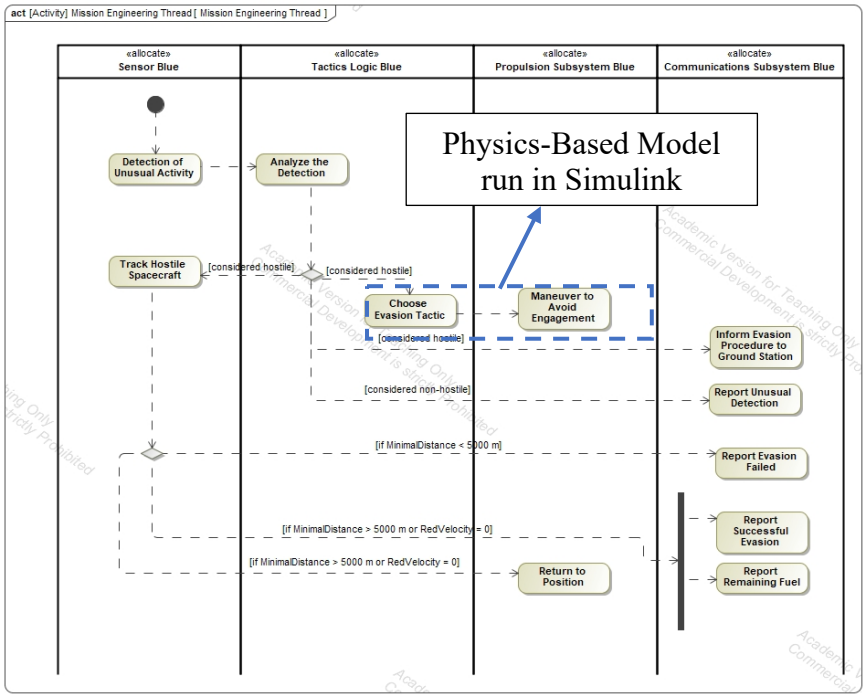

Figure 16. Tasks covered in the Simulink model.

B. SIMULINK MODEL OVERVIEW

The Simulink model this study uses represents two systems: one attacker (ASAT-Red) and one defender (SoI-Blue). The dynamics of both systems are modeled using a state transition matrix developed from the Clohessy-Wiltshire (CW) equations. According to [39], the model's core objective is to allow for rapid and sudden changes in spacecraft velocity; keeping a high degree of fidelity for the assessment of real-life engagements, this simulator is a starting point for tactics assessment. Its development did not consider use within an executable SysML model. Still, a modular approach was taken, which facilitated its integration with some minor modifications. Figure 17 conveys the model arrangement, the blocks in the descriptive model that send the inputs to their counterparts in Simulink, and the part that send outputs to MATLAB workspace.

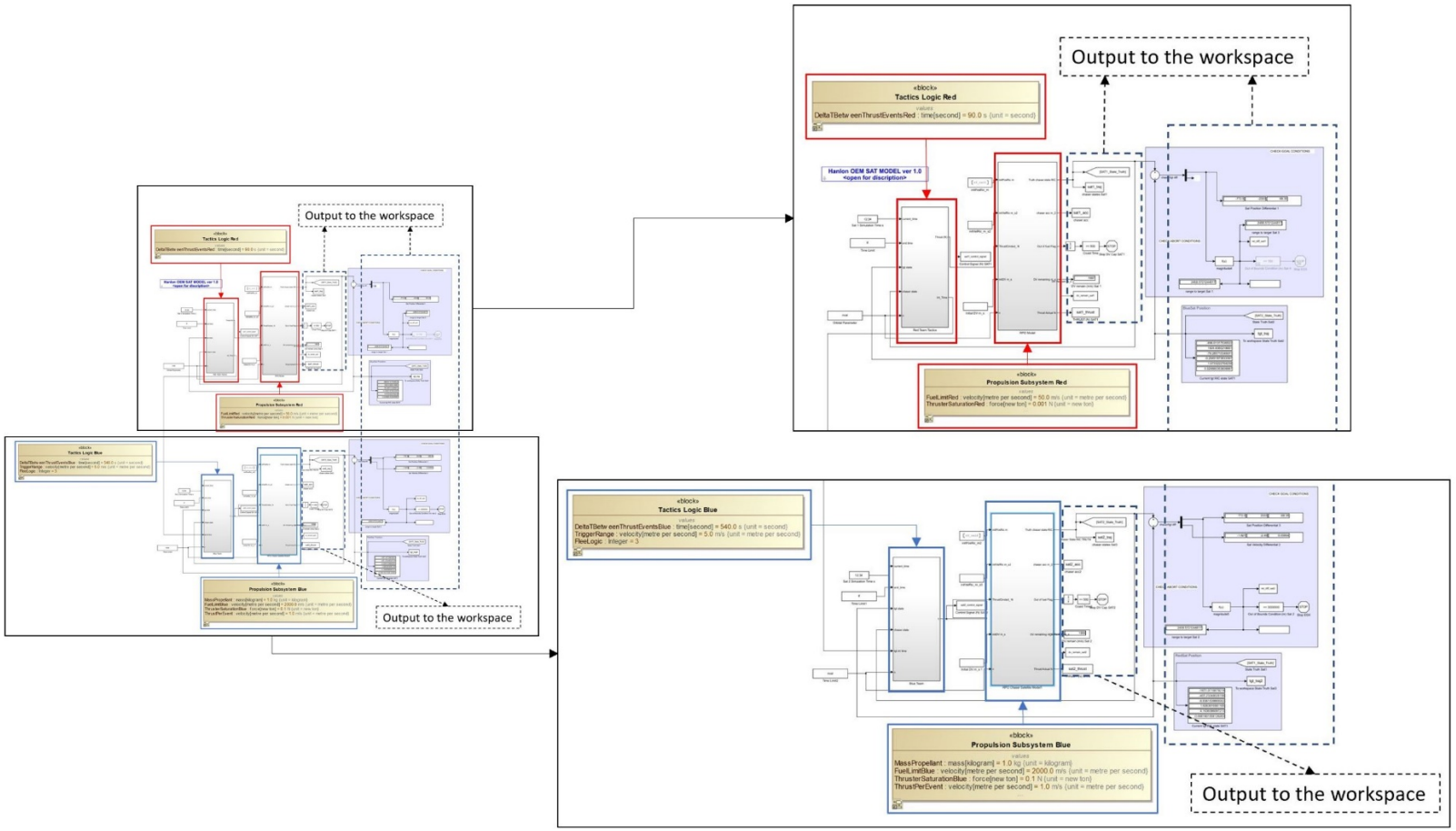

Figure 17. Illustration of interfaces existing between the CSM and Simulink models. Adapted from [12].

The radial, in-track, cross-track (RIC) trajectory-based reference frame is the coordinate system adopted due to its suitability to RPO maneuvers. According to [39], the CW equations describe the spacecraft's relative motion to a constant point in the coordinate system. This chosen constant point is the initial defender's position, which remains stationary in the RIC frame [39]. However, the inertial reference frame disregards any defender's maneuvers [39].

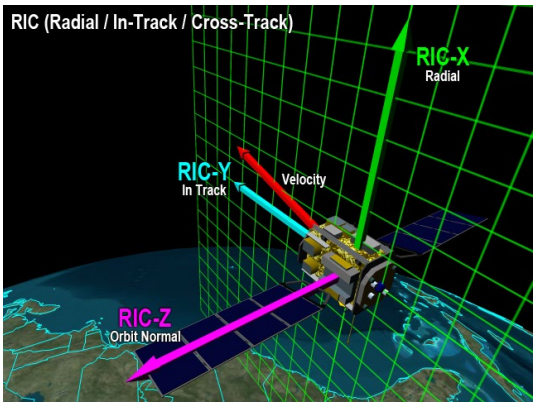

Figure 18. RIC trajectory reference frame. Source: [40].

The model uses a so-called "low-fidelity" propagator to estimate a spacecraft's position considering orbital mechanics and their maneuvers [12]. This propagator only considers the Earth's gravity and the velocity changes generated by the propulsion subsystem [12]. Nevertheless, [12] demonstrated that the model's performance over a short time span is similar to high precision propagators such as the System Tool Kit's Astrogator propagator.

The simulation starts with both spacecraft a few kilometers apart. This distance is the initial condition of the problem and may indicate a sensor range, for example. Hanlon [12] tested various directions considering various relative positions between defender and attacker to develop a tactic that results in 30–50% savings in fuel consumption relative to the aggressor. In this model, both spacecraft were considered nearly identical, which is unrealistic but necessary for the comparison purpose.

In the model, the satellites' maneuvers around each other are decided by their tactics block. The tactics blocks take the current SoI and ASAT configuration and establish a response considering the ASAT state, the own state, and the simulation remaining time [39]. At the same time, the outputs are thrust commands [39]. While the attacker tries to intercept the defender on the most effective trajectory considering the engagement's remaining time, the defender uses the aforementioned fuel-saving approach.

The RPO block transforms the tactics block's commands into thrust commands for velocity changes; it tracks the fuel consumed and the spacecraft's position and velocity relative to the target's initial position [39]. Besides, according to [39], "it calculates what the state is in the next time step." As usual, Simulink sends the data generated to the MATLAB workspace after each run. Initially, the user could visualize the simulation outputs for each engagement through a series of graphics produced in the MATLAB environment. This aspect changed in this thesis due to the factors considered relevant in this mission analysis and to take better advantage of the executable model developed. Chapter IV discusses the presentation of results in the executable model, while more information about this Simulink model can be found in [12], [39].

C. ORIGINAL MODEL MODIFICATIONS

The Simulink model described in [12] is used in this thesis essentially as a black box. It receives some systems' parameters as inputs and returns MoPs and graphical representations as outputs. The few modifications implemented do not affect the already proven high-fidelity of the simulation. Instead, they allow for modification of the unrealistic assumption that both spacecraft are identical, which increases the level of realism.

The first modification concerns the mass of the spacecraft. Hanlon assumed that both spacecraft have the same mass equal to 1 kilogram [12]. This characteristic is adjustable from the descriptive model, and it is one of the inputs when the model initializes. For example, SoI mass is ten times greater than the ASAT mass in the Chapter V demonstration. Other inputs further differentiate the attacker and the defender, helping to create more representational engagements.

The second modification is in the storage and presentation of the results. Originally, a *.mat* file stores all data produced in the simulation allowing the necessary manipulation of this data for analysis later. Although all the original data generated after the simulation is still available in the shared workspace, only the satellites' minimum distance and their fuel consumption are displayed in the GUI for requirements verification. Also, the GUI can produce a graph that shows the trajectory of both spacecraft during the engagement. For further mission analysis, the GUI also includes two animations showing the kinematics of the engagement and the thrust of the propulsion subsystem.

The meaning of simulation time is another aspect that deserves a discussion. When executing a SysML diagram in CSM, the simulation time shown in the CSM simulation console is not representing the simulation time considered inside the physics-based model. The simulation time is an input, and all the simulation occurs without user intervention. This limitation is not a CSM or an integration issue. Instead, it is a consequence of the model's design. Figure 19 conveys a high-level data flowchart view of the simulation showing inputs and outputs adapted from [39]. The next chapter covers this data exchange in more detail.

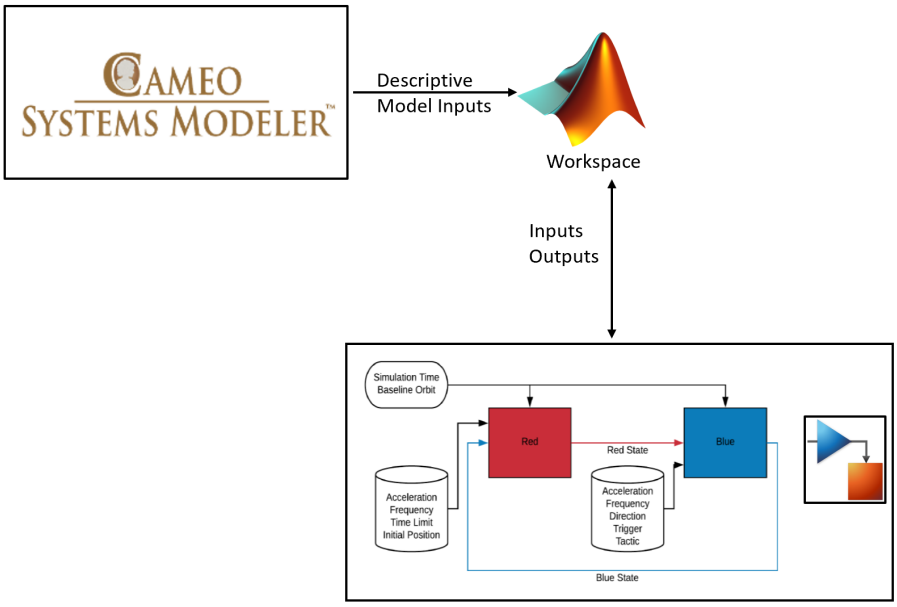

Figure 19. High-level view of the data flow between two environments.

IV. CSM/SIMULINK INTEGRATION

This chapter explains the design of the executable model for COE mission analysis using the descriptive and physics-based models presented in the previous two chapters. It also covers the technical aspects and a few best practices related to the integration of and information flow between CSM and MATLAB/Simulink to develop an executable model using these tools. This chapter's primary goal is to provide all the information necessary for integrating and using these tools, allowing effortless development of executable models dedicated to mission analysis in future works.

A. DATA EXCHANGE

The first step to enable this integration is to install the CST plug-in that allows external evaluators, as mentioned before. Also, MATLAB integration as a tool must occur to CSM to call and use it in CST [41]. This integration does not mean that CST will use MATLAB for every calculation in the model, since the built-in CST math engine is still available and is the default evaluator. Instead, it means that CST will correctly interpret MATLAB commands when they appear within the model.

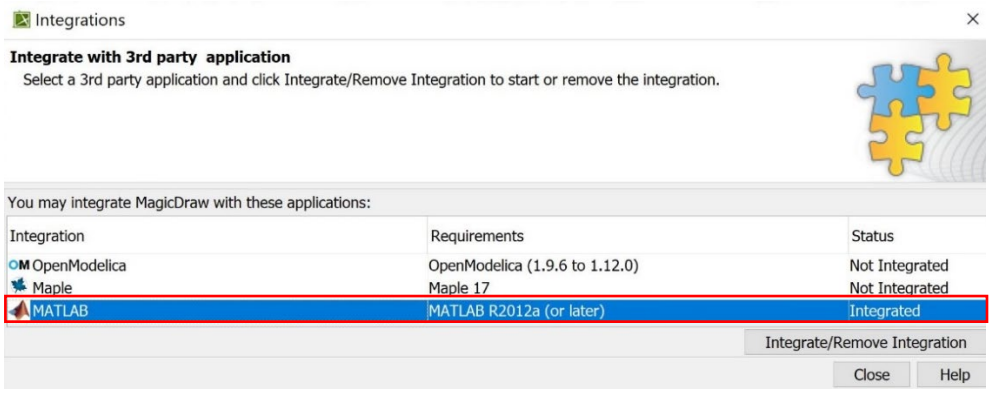

Figure 20. MATLAB integrated with CST.

The design of the Simulink legacy model in question did not consider this kind of integration. So, the best alternative in this case is its use as a black box nested in the descriptive model. The key to a successful integration, in this case, is to assure that both

tools share the same MATLAB workspace. This is a better approach, since it minimizes interferences in the MATLAB code/Simulink model design. Cameo Systems Modeler creates its own hidden workspace when it uses MATLAB as an external evaluator. This hidden workspace will be disabled by calling the *kill matlab* command in the Simulation Console panel in the Simulation window provided by CST [42], [43]. Then, it is necessary to initiate MATLAB and convert the currently running session into a shared session. This sharing can be done by calling the *matlab.engine.shareEngine* command in MATLAB Command Window [43], [44]. After that, the descriptive model inside CSM is the one ruling any other model developed in the Simulink environment, and all the other commands will be called from CSM. Figure 21 shows a BDD that conveys the executable model's structure.

MATLAB/Simulink runs in the second plane to support the descriptive model, and any native MATLAB functions are automatically recognized. Simulink models, MATLAB scripts written as a function, or any other user-defined functions used in the executable model after the integration will be stored either in the CSM directory or in a path indicated as the Current Folder window in MATLAB. Otherwise, CST will not recognize these functions or models.

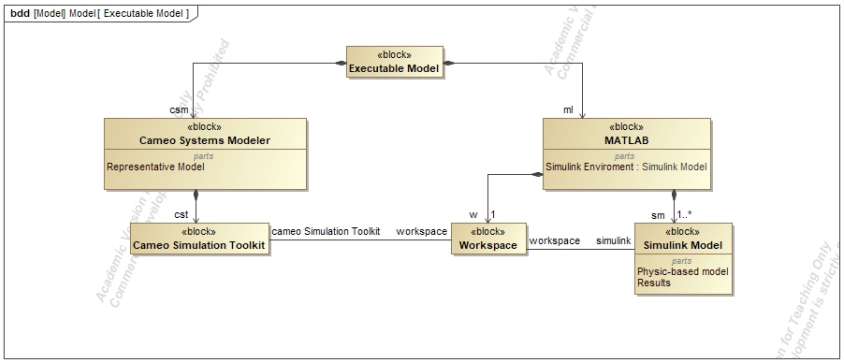

Figure 21. BDD of the executable model.

Now that both tools share a common place to write and read data, it is essential to know the options to include or the variables to delete in the shared workspace. There are three ways to do that: calling commands through the CST Simulation Console, using

activity diagrams with opaque actions, or using opaque behaviors inside states in a state machine diagram (Figure 22). An opaque action is an action that has a body (coding lines) and language (programming language) properties whose functionality is not specified within UML [45], [38]. These two properties allow writing and executing mathematical expressions in various programming languages [23], [46], making this the preferable way to execute MATLAB commands and functions and even the Simulink model. Additionally, activity diagrams are excellent to represent complex control logic and the most effective analysis tool in SysML [23]. Activity diagrams can also be incorporated as a behavior in a state machine diagram. For this reason, the descriptive model uses several activity diagrams to facilitate the information exchange. The other option is the opaque behavior which is another state's behavior type with the same structure as the opaque actions and which executes MATLAB functions and commands in the same way. Figure 23 conveys an internal block diagram of the data flow within the executable model.

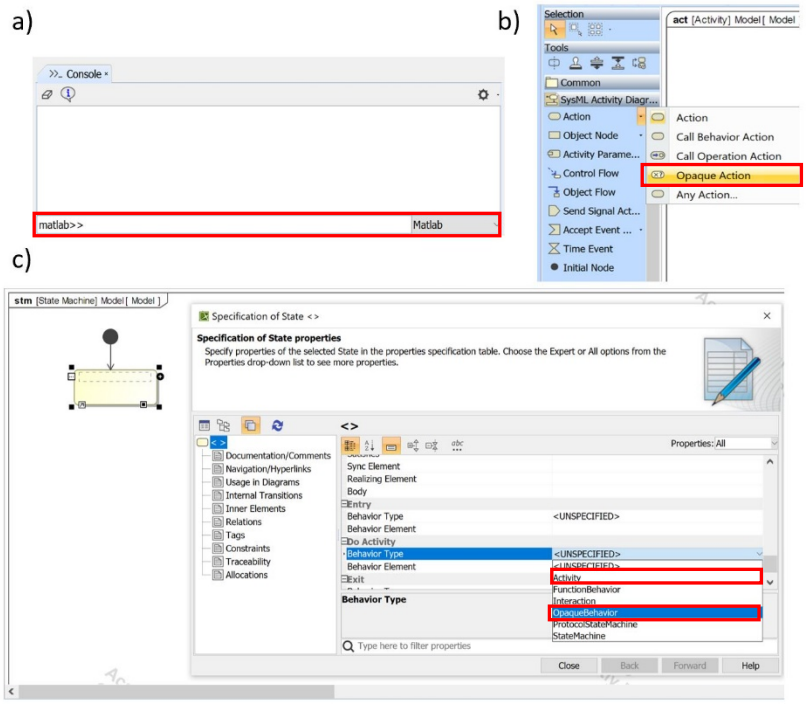

Figure 22. Three ways to send data from CST to MATLAB:
a) CST Console Panel; b) Opaque Actions;
c) Behavior Type: Activity or Opaque Behavior.

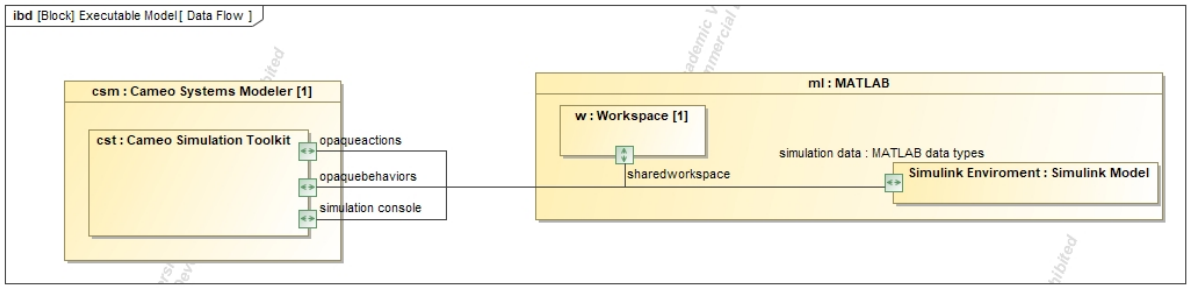

Figure 23. Internal data block showing data flow.

In addition to having a shared workspace, two other aspects are crucial to making the integration successful. The first one is ensuring that CSM gives the external evaluator enough time to run the simulation, calculate the requested outputs, and write them in the shared workspace to use them in CSM. This technical detail cannot be neglected; otherwise, the model will not run properly. It is only necessary to increase the timeout for external solvers in the CSM simulation options menu to address this issue [47]. This adjustment depends on the complexity and intensity of the calculations expected for the external solver. In this executable model, the timeout is set for six minutes, which is more than enough for the simulator to produce the requested outputs even for 24-hour period simulations (Figure 24).

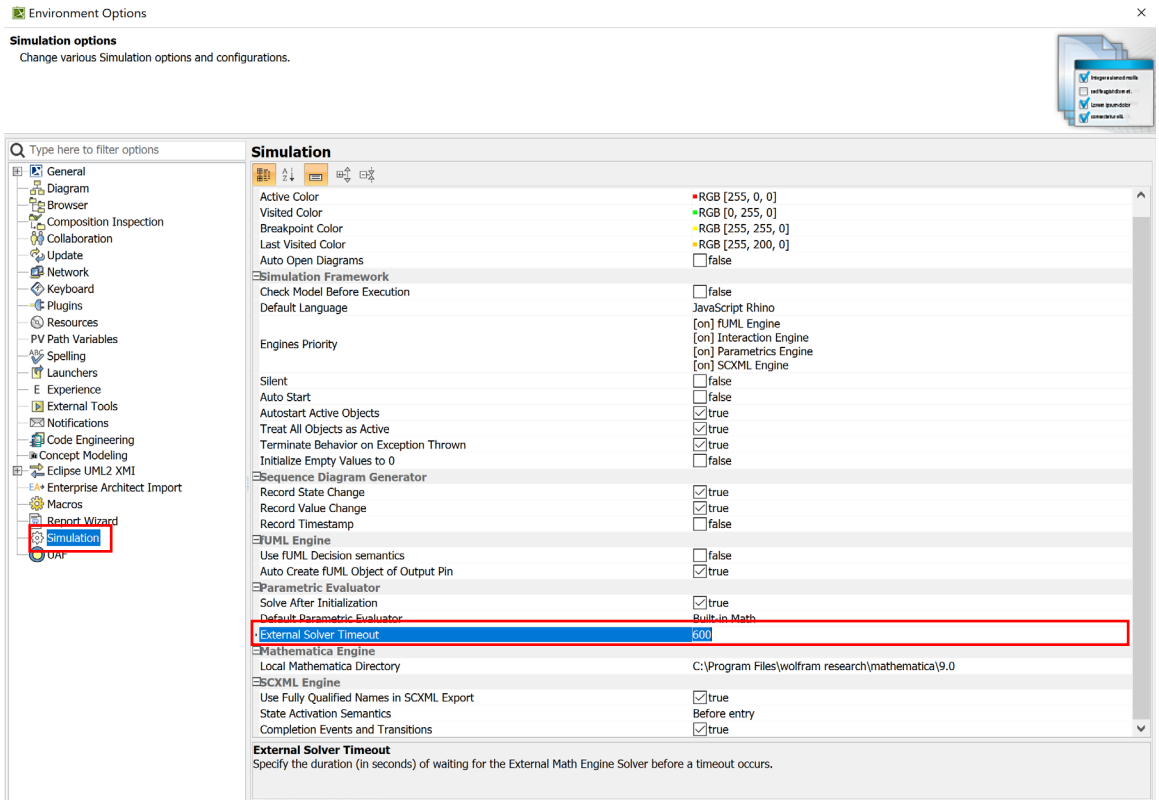

Figure 24. External Solver Timeout.

The second aspect is to guarantee that MATLAB receives the system data in the correct format and name. Usually, integer or real numbers represent the value properties describing aspects of the system. Besides, CSM does not support value properties written in a vector or matrix format, which are needed when working in the MATLAB/Simulink environment. Due to this, adjustments may be required using the opaque actions to write the inputs properly in the MATLAB workspace. Figure 25 shows an example of it in the descriptive model. The initial position and velocity of both spacecraft are necessary as inputs to start the simulation, and both data must be written as 1x3 matrices. To address that, an activity diagram containing an opaque action is embedded in this block. In fact, every block representing a subsystem also embeds a similar activity diagram to send inputs to the shared workspace, whether in the matrix form or not. This is necessary, since activity diagrams can only read the value properties of the block that serves as a context for them. This minor limitation must be considered when deciding where to declare properties of the system and its subsystems.

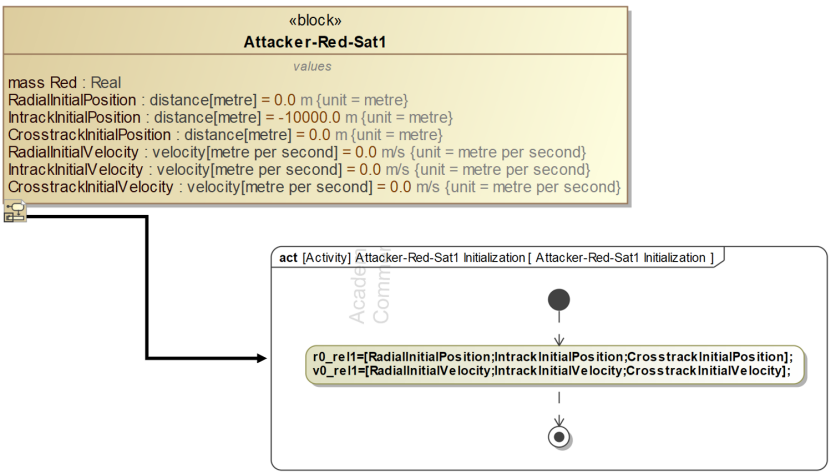

Figure 25. Example of variable adjustment using opaque actions.

The naming of variables is another point worth mentioning. Descriptive models give the modeler the opportunity to declare systems' properties in a more readable way than is usually provided in programming languages. This fact makes these models more understandable for all stakeholders, especially non-technical ones, than the hundreds of lines of code associated with a physics-based model like the one in question. Nevertheless, these opaque actions must match the value properties names and the simulator's variables names to ensure that both models find the data they need in the shared workspace.

B. GRAPHICAL USER INTERFACE

As stated before, a GUI makes the executable model more user-friendly and the analysis of the system and its mission easier. The first step to build this GUI is creating a state machine diagram nested inside a block to be its classifier behavior. A block can have several diagrams representing its behavior. However, only one of them can be the block's classifier behavior. In this example, the COE Mission block is in the top-level of the mission structure derived from the vignette. For this reason, the state machine diagram developed is nested inside this block and used as its classified behavior.

In a state machine diagram, a state can have three kinds of behaviors: *entry* behaviors, *do* behaviors, and *exit* behaviors. An *entry* behavior is the first behavior executed whenever the state is entered, and an *exit* behavior is executed before a state is

exited [23], [33]. A *do* behavior is executed immediately after the entry behavior [23]. Activity diagrams can represent *entry*, *do*, and *exit* behaviors, which facilitates the execution of MATLAB commands within a specific state. So, each state in the diagram has at least one activity diagram representing behavior. Figure 26 conveys the arrangement in tiers of the diagrams. In the containment tree, all these activity diagrams are hierarchically below the COE Mission block for model organization purposes. This is not required, however, and the model's organization can include a library to store them all to facilitate their reuse for a descriptive model integrated with more than one physics-based model.

Triggers are one of the options to cause the transition between states [33]. A Signal event is one kind of trigger that represent that an asynchronous signal has arrived [33]. Since the state machine diagram's main goal is to support the mission analysis GUI, signals are the best option to represent the user's commands to the model. This procedure completes the design of the state machine diagram contributing a lot to make the model easier to use as an analysis tool that keeps MATLAB totally in the second plane.

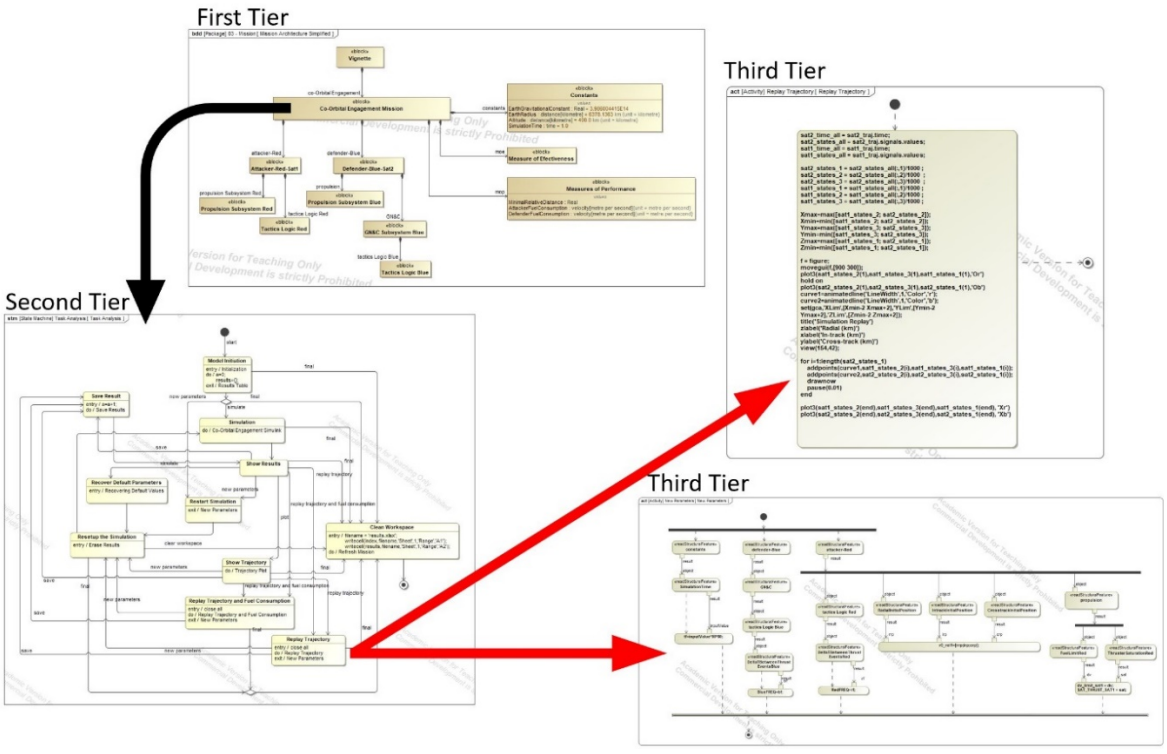

Figure 26. Relationship between the diagrams.

The next step is the design of the user interface diagram. To create this diagram, it is crucial to understand which parts of the model will be executed, which will serve as inputs, and which results are important to display. Having the BDD of the mission's structure as a guide, the main frame called Mission Analysis in the UI diagram represents the Vignette block. Within this container, there are several panels representing the COE Mission block and its parts. Every button in the GUI is related to one signal event in the state machine diagram that is the classifier behavior of the COE Mission block. Both SoI and ASAT parameters shown in the GUI can be manipulated by users in their analysis either through sliders or by typing new values into the text fields reserved for them. The simulation time in the GUI is the one considered by the physics-based model and can be manipulated by the user as well. The three MoPs are displayed for user assessment of success or failure of each engagement run. Figure 27 shows the correlation between UI entities and the mission structure parts.

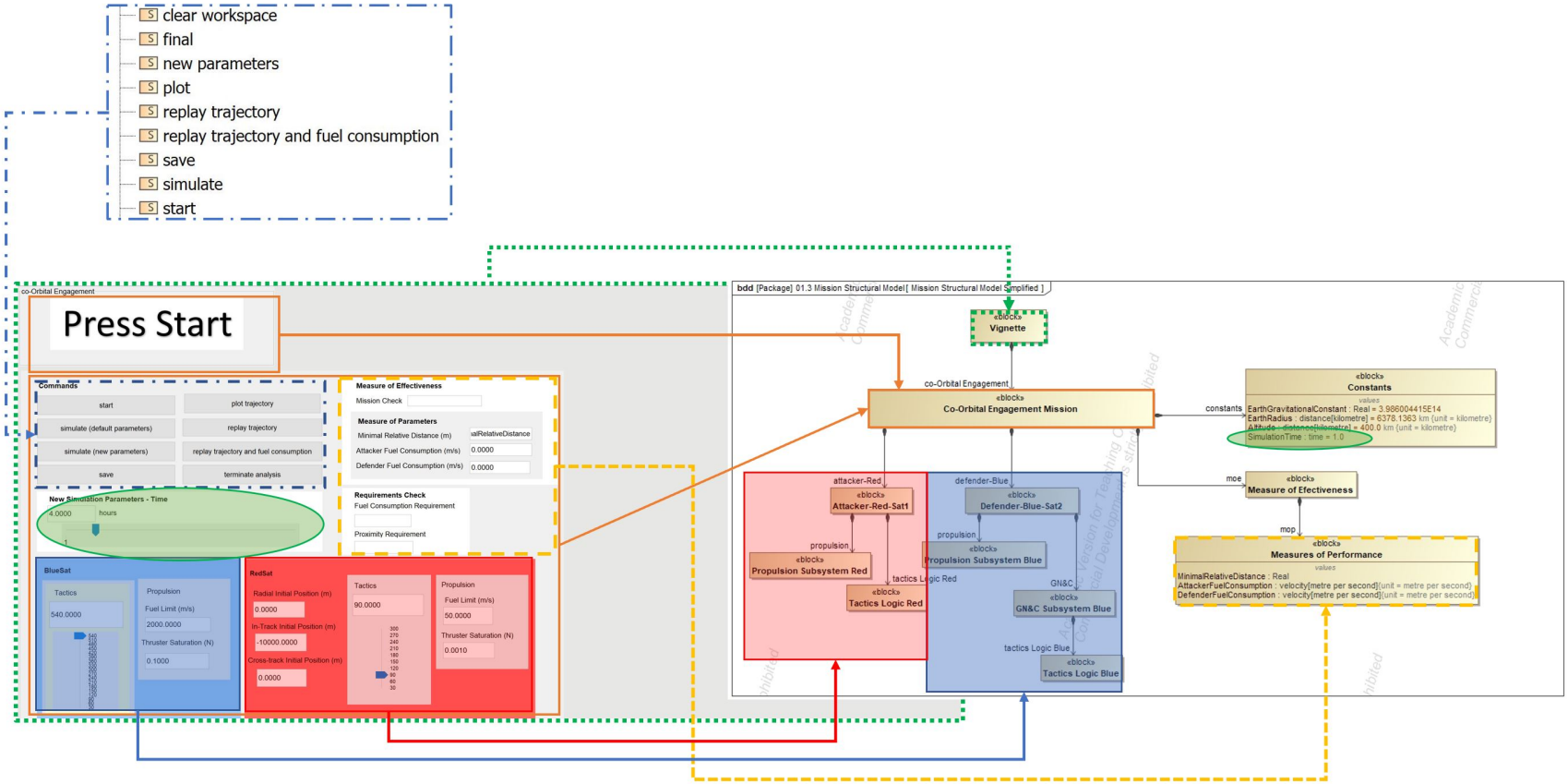

Figure 27. Correlation between the UI entities and mission's structure parts.

Some characteristics of the GUI deserve more discussion, since there are ways to overcome the CSM UI diagram's limitation, and ways to leverage CSM and MATLAB as integrated tools or decisions made to facilitate the exploration of the design space. The first of them is the image that shows the simulation status. Located in the upper left corner of the GUI, this image changes according to the stage of the analysis. Even though CSM animates the state machine diagram to show the same information, follow that in the diagram when using a GUI is not practical.

On the other hand, the decision to include this user visual aid has another purpose. CSM does not have any option that allows the locking of a button to be conditioned according to the current state of the system. This can lead to situations where the selection of an option by the user does nothing simply because it does not correspond to the signal event necessary to trigger a transition between states. There are seven different messages indicating the states where the user needs to press a button to trigger the transition (Figure 28). States that do not need user intervention to change have no message. Another option is to use a similar visual aid with instructions about the options the user has in each one of the stages. In both cases, the element used is the image switcher, which is a CST feature that allows the representation of a specific stage through an image in the GUI.

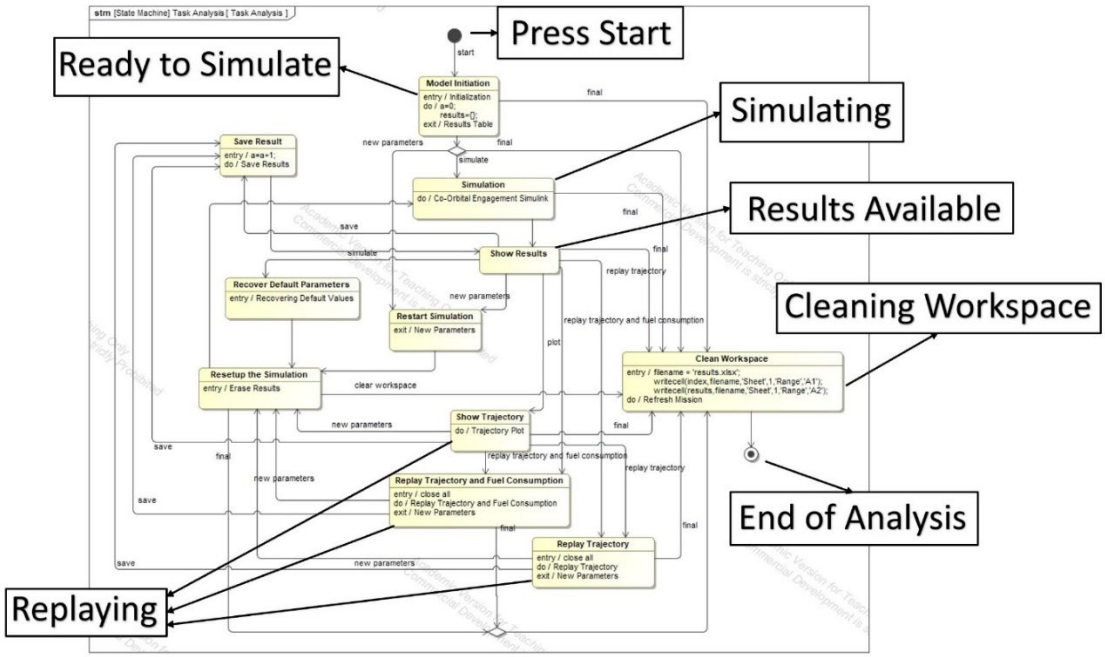

Figure 28. Messages associated with each state in the GUI state machine diagram.

The second noteworthy aspect is the updating of a block's value properties during the analysis. A block's value property can have a defined type and a default value. In this example, the default values in each of the BDDs correspond to the system's current status. Besides, at the beginning of the analysis, the values displayed in the GUI, when they exist, are these default values.

For exploring the design space and the impacts of changes in one or more system parameters, however, the analysis must allow for them to change quickly and without compromising the consistency within the model. The systems and mission's parameters considered essential for this analysis are available in the GUI for user modification. Changes in the system's parameter using one of the sliders or the GUI text fields only affect the blocks' instance created by in CST for this analysis, updating their value property with these inputs. So, since these modifications in the GUI do not change the default values in the BDDs, they allow the assessment of different designs without introducing inconsistencies in the model as desired. From a data consistency perspective, this aspect is

excellent. On the other hand, it raises an integration issue. The shared workspace does not receive any updates from GUI automatically.

In the initialization stage, the default values are sent to the shared workspace creating the input variables needed for the simulator to be ready to run. To modify these values, an activity diagram that uses both UML actions and MATLAB commands is necessary. First, the UML action called *readStructuralFeatureAction* retrieves the values set by the user in the GUI. After that, these values overwrite the variables already declared in the shared workspace through opaque actions using MATLAB commands. The necessity of this step accounts for the two different buttons to start the simulation. The first one runs the simulator using the values already available in the workspace, and the second one overwrites the variables in the workspace with the new user-defined new values. A similar procedure is required for the information generated by the simulator, since that information is not automatically updated in the descriptive model either. MoPs are an example of this kind of information.

The three MoPs shown in the GUI have no values before the first run of the simulator. Based on the simulation's results, an opaque action calculates the desired MoPs using MATLAB commands. Then these values are updated in the blocks' instances created by CST through the *addStructuralFeatureValueAction*. The GUI always shows the updated values for the user. Figure 29 conveys how the information update process takes place in the executable model.

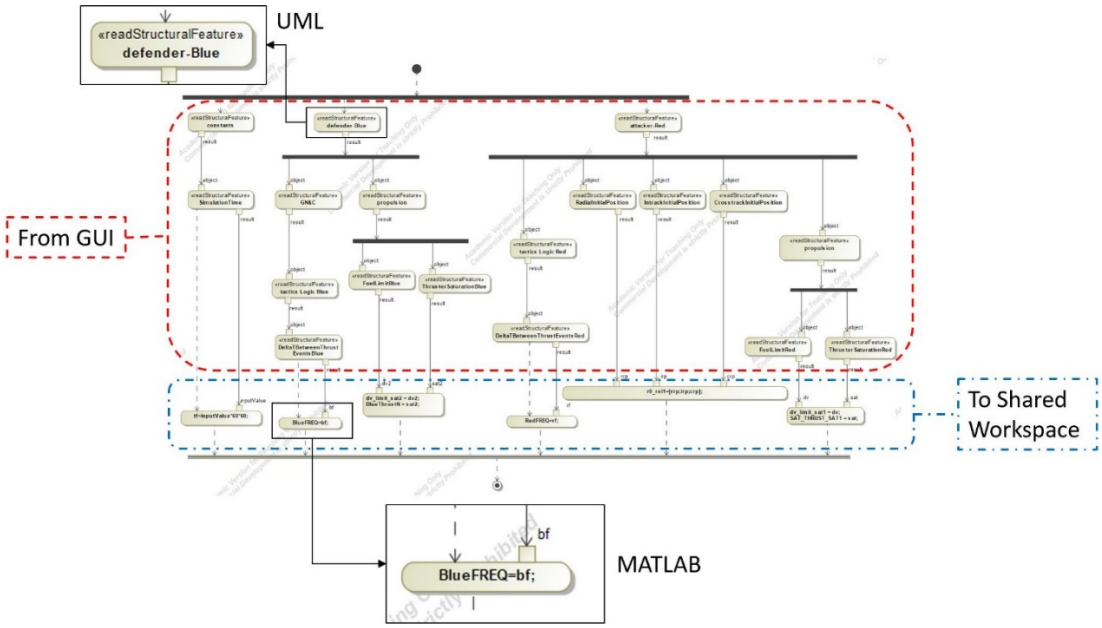

Figure 29. Information update flows from the descriptive model to the shared workspace.

The third aspect concerns how to take advantage of the fact that MATLAB is the external evaluator for the executable model. The fact that all native MATLAB functions are available opens new possibilities for visualization of data and the ability to use graphical tools in the analysis of a descriptive model, which would otherwise, not be available. There are four buttons in the GUI that demonstrate these advantages. The first one is a save button that saves all the inputs and outputs of the run in an Excel spreadsheet for further analysis and comparisons. The other three buttons utilize some MATLAB graphical functions to recreate and display both ASAT and SoI trajectories. This graphical feature was not originally embedded in the Simulink model. However, the only data necessary to generate them were the outputs available in the shared workspace. This feature is significant for tactics assessment since it can show operational stakeholders how technical parameters influence specific aspects of the mission.

One GUI limitation is that it cannot display figures generated in the MATLAB domain. The major inconvenience this limitation brings is that the figures can pop up in front of buttons or other important information on the GUI, or they may even stay hidden

behind the CSM window. Fortunately, overcoming this issue is quite simple. It is just necessary to use the proper MATLAB commands to ensure that the figure pops up in a convenient space on the screen. The GUI has a specific place reserved for presenting these graphics. Figure 30 shows how the MATLAB figure fits in the screen.

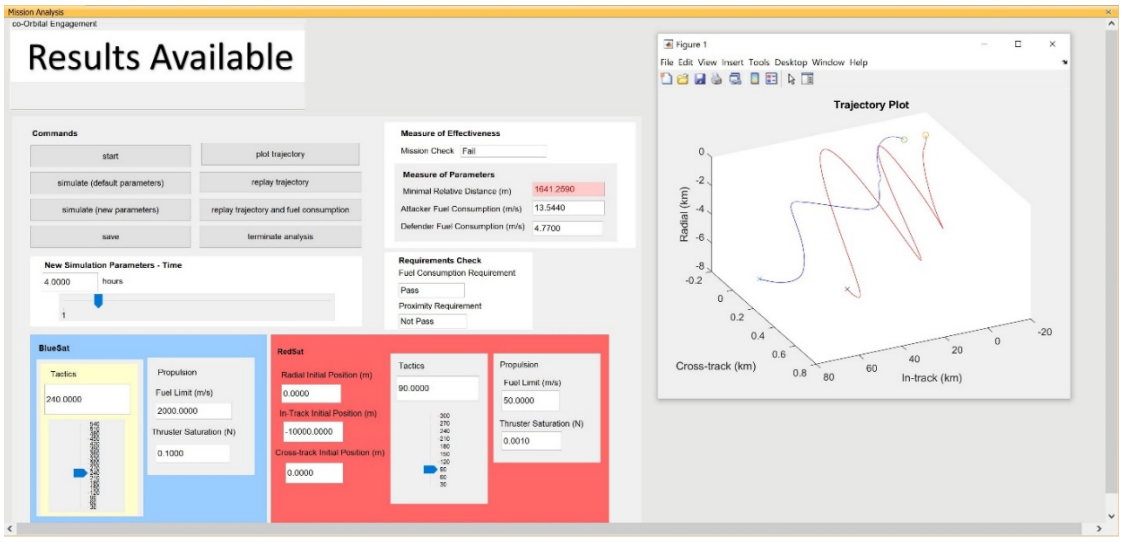

Figure 30. GUI and MATLAB graphs.

The last step in setting the GUI is to create a simulation configuration element in a Simulation Configuration Diagram to customize some options for running the simulation in the context of the Vignette block. The simulation configuration element has the Vignette block as the execution target and is the responsible of starting the GUI when the simulation runs. The same diagram nests the image switcher representing the states of the state machine diagram that rules the GUI.

This step ends the necessary procedures for using the GUI to control the executable model. From the GUI, the user can control all the available simulation parameters. It is possible to improve the GUI to give the user other options. However, for the desired proof of concept, the options available are enough. The main limitation of the designed GUI is that it only runs one case at a time. If running of multiple cases is necessary, another CSM feature can handle it: the instance table diagram.

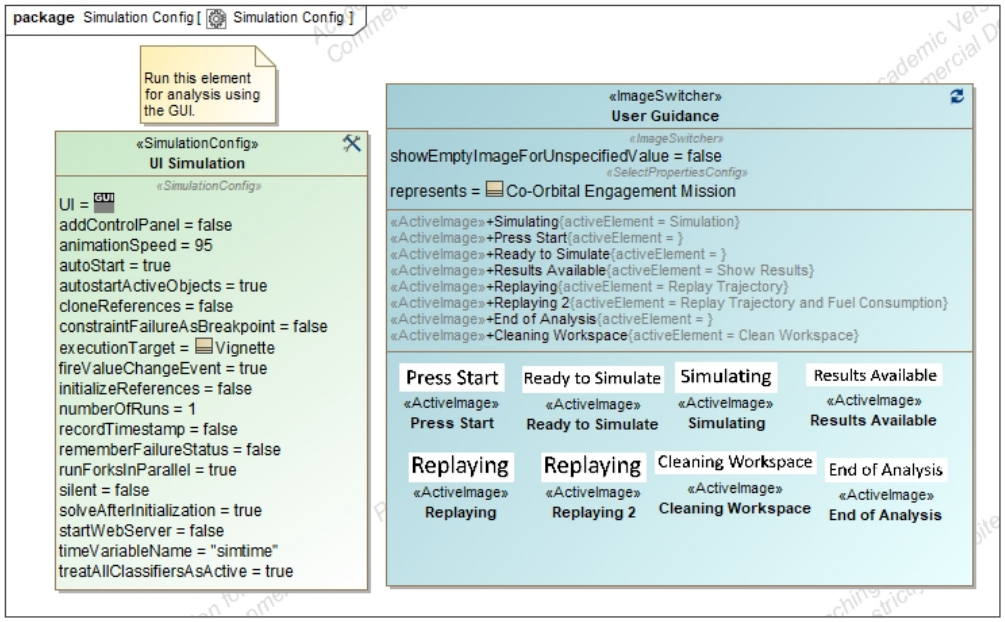

Figure 31. Simulation configuration diagram.

C. THE INSTANCE TABLE

Despite all GUI benefits, a faster analysis of the design space requires the use of the instance table diagram as mentioned in Chapter II. This diagram allows the management of multiple instances of the same mission in a spreadsheet-like format that enables modifications in the studied input parameters. Both the GUI and the instance table run the Simulink model to calculate and to show the user the results of the ASAT COE attempt. Nonetheless, there are a few modifications needed in the COE block to make the analysis from the table possible.

First, a new BDD is created replicating the mission structure shown in Figure 8 in a new package called Trade Studies Analysis. This BDD reuses the same blocks defined before except for the COE Mission block, the modifications of which are explained in the next paragraph. It means that all value properties that serve as input for the simulator, and the activity diagrams responsible for sending and recovering data from the shared workspace are the same.

The second modification is in the COE block classifier behavior that can no longer be the state machine diagram. This modification is necessary due to the design of the state

machine diagram that aims the user commands to transit between states as mentioned previously. Since the simulations from the table have no user intervention, another instance of this block is created, and the classifier behavior of this instance is changed for the activity diagram responsible for running the simulator and calculating the outputs. This activity diagram was originally used as a *do* behavior in the Simulation state of the state machine diagram. All the other activity diagrams representing all *entry*, *do*, and *exit* behaviors are not used in the instance table. This block was renamed as COE Trade Studies.

The last modification aims to ensure that the *readStructuralFeatureAction* in this activity diagram is retrieving the value of the correct structural feature. This is only necessary for the newly created COE Trade Studies block. This modification can be easily done by dragging and dropping the part property of the new block into the corresponding feature action.

In the instance table, the COE Trade Studies block is set as the classifier. Each row represents an instance of this block and the columns can show any value property from the block or its parts. The user can customize what he or she thinks is valuable for the analysis. Figure 32 shows the produced instance table displaying the initial position of the ASAT, how fast both SoI and ASAT maneuver, the SoI fuel consumption, how close the spacecraft gets to each other, and the mission status based on the operational requirements.

#	Name	Attacker-Red-Sa : di	Attacker-Red-Sat : dista	Attacker-Red-Sat : dista	Attacker-Red-S Red.DeltaTBetwe : time	Defender-Blue- L Blue.DeltaTBetwe : time	moe2.mop.Defer : velocity[m	moe2.mop.Minim :	moe2.MissionSucc : String
1	co-orbital Engagement Trade Studies	-4618.8 m	-4618.8 m	-4618.8 m	90 s	540 s	0 m/s	7999.9963	Success
2	co-orbital Engagement Trade Studies1	-5656.85 m	-5656.85 m	0 m	90 s	540 s	0.27 m/s	4762.6351	Fail
3	co-orbital Engagement Trade Studies2	-4618.8 m	-4618.8 m	4618.802 m	90 s	540 s	0 m/s	7999.9974	Success
4	co-orbital Engagement Trade Studies3	-5656.85 m	0 m	-5656.85 m	90 s	540 s	0 m/s	7999.994	Success
5	co-orbital Engagement Trade Studies4	-8000 m	0 m	0 m	90 s	540 s	2.34 m/s	6.437	Fail
6	co-orbital Engagement Trade Studies5	-5656.85 m	0 m	5656.854 m	90 s	540 s	0 m/s	7999.9968	Success
7	co-orbital Engagement Trade Studies6	-4618.8 m	4618.802 m	-4618.8 m	90 s	540 s	0 m/s	7999.9974	Success
8	co-orbital Engagement Trade Studies7	-5656.85 m	5656.854 m	0 m	90 s	540 s	0.27 m/s	4762.6389	Fail
9	co-orbital Engagement Trade Studies8	-4618.8 m	4618.802 m	4618.802 m	90 s	540 s	0 m/s	7999.9986	Success
10	co-orbital Engagement Trade Studies9	0 m	-5656.85 m	-5656.85 m	90 s	540 s	0 m/s	7999.994	Success
11	co-orbital Engagement Trade Studies10	0 m	-8000 m	0 m	90 s	540 s	2.34 m/s	4.1171	Fail
12	co-orbital Engagement Trade Studies11	0 m	-5656.85 m	5656.854 m	90 s	540 s	0 m/s	7999.9968	Success
13	co-orbital Engagement Trade Studies12	0 m	0 m	-8000 m	90 s	540 s	0 m/s	8000	Success
14	co-orbital Engagement Trade Studies13	0 m	0 m	8000 m	90 s	540 s	0 m/s	8000	Success
15	co-orbital Engagement Trade Studies14	0 m	5656.854 m	-5656.85 m	90 s	540 s	0 m/s	7999.9968	Success
16	co-orbital Engagement Trade Studies15	0 m	8000 m	0 m	90 s	540 s	2.34 m/s	4.1171	Fail
17	co-orbital Engagement Trade Studies16	0 m	5656.854 m	5656.854 m	90 s	540 s	0 m/s	7999.9996	Success
18	co-orbital Engagement Trade Studies17	4618.802 m	-4618.8 m	-4618.8 m	90 s	540 s	0 m/s	7999.9974	Success
19	co-orbital Engagement Trade Studies18	5656.854 m	-5656.85 m	0 m	90 s	540 s	0.27 m/s	4762.6354	Fail
20	co-orbital Engagement Trade Studies19	4618.802 m	-4618.8 m	4618.802 m	90 s	540 s	0 m/s	7999.9986	Success
21	co-orbital Engagement Trade Studies20	5656.854 m	0 m	-5656.85 m	90 s	540 s	0 m/s	7999.9968	Success
22	co-orbital Engagement Trade Studies21	8000 m	0 m	0 m	90 s	540 s	2.34 m/s	6.437	Fail
23	co-orbital Engagement Trade Studies22	5656.854 m	0 m	5656.854 m	90 s	540 s	0 m/s	7999.9996	Success
24	co-orbital Engagement Trade Studies23	4618.802 m	4618.802 m	-4618.8 m	90 s	540 s	0 m/s	7999.9986	Success
25	co-orbital Engagement Trade Studies24	5656.854 m	5656.854 m	0 m	90 s	540 s	0.27 m/s	4762.6392	Fail
26	co-orbital Engagement Trade Studies25	4618.802 m	4618.802 m	4618.802 m	90 s	540 s	0 m/s	7999.9997	Success
27	co-orbital Engagement Trade Studies26	-5773.5 m	-5773.5 m	-5773.5 m	90 s	540 s	0 m/s	9999.9953	Success
28	co-orbital Engagement Trade Studies27	-7071.07 m	-7071.07 m	0 m	90 s	540 s	0.27 m/s	6210.9025	Fail
29	co-orbital Engagement Trade Studies28	-5773.5 m	-5773.5 m	5773.503 m	90 s	540 s	0 m/s	9999.9971	Success
30	co-orbital Engagement Trade Studies29	-7071.07 m	0 m	-7071.07 m	90 s	540 s	0 m/s	10000.0031	Success
31	co-orbital Engagement Trade Studies30	-10000 m	0 m	0 m	90 s	540 s	2.34 m/s	5.584	Fail
32	co-orbital Engagement Trade Studies31	-7071.07 m	0 m	7071.068 m	90 s	540 s	0 m/s	10000.0017	Success
33	co-orbital Engagement Trade Studies32	-5773.5 m	5773.503 m	-5773.5 m	90 s	540 s	0 m/s	9999.9971	Success
34	co-orbital Engagement Trade Studies33	-7071.07 m	7071.068 m	0 m	90 s	540 s	0.27 m/s	6210.9006	Fail
35	co-orbital Engagement Trade Studies34	-5773.5 m	5773.503 m	5773.503 m	90 s	540 s	0 m/s	9999.9988	Success
36	co-orbital Engagement Trade Studies35	0 m	-7071.07 m	-7071.07 m	90 s	540 s	0 m/s	10000.0031	Success
37	co-orbital Engagement Trade Studies36	0 m	-10000 m	0 m	90 s	540 s	2.34 m/s	1.9071	Fail
38	co-orbital Engagement Trade Studies37	0 m	-7071.07 m	7071.068 m	90 s	540 s	0 m/s	10000.0017	Success
39	co-orbital Engagement Trade Studies38	0 m	0 m	-10000 m	90 s	540 s	0 m/s	10000	Success
40	co-orbital Engagement Trade Studies39	0 m	0 m	10000 m	90 s	540 s	0 m/s	10000	Success
41	co-orbital Engagement Trade Studies40	0 m	7071.068 m	-7071.07 m	90 s	540 s	0 m/s	10000.0017	Success
42	co-orbital Engagement Trade Studies41	0 m	10000 m	0 m	90 s	540 s	2.34 m/s	1.9071	Fail
43	co-orbital Engagement Trade Studies42	0 m	7071.068 m	7071.068 m	90 s	540 s	0 m/s	10000.0003	Success

Figure 32. Example of the instance table.

The setting of the instance table completes the design of the executable model and the analysis design. So, it marks the end of the third step of the process shown in Figure 3. Chapter V presents a demonstration using both the Instance Table and the GUI for the analysis of the effect of one subsystem parameter in the COE mission.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DEMONSTRATION OF MODEL EXECUTION

This chapter presents an example of a tradeoff study performed to illustrate the process of verifying the suitability of choosing specific parameters for one of the subsystems. The chapter starts with the motivation for choosing some specific parameters to illustrate how to use the executable model in early design phases to simultaneously support the mission and system assessment. Then, it presents an example of how a batch-run can be executed to identify critical scenarios where the default value does not meet the operational requirements (using the instance table). Next, these failure cases are further analyzed for changes in this specific parameter, also using the instance table. Finally, this chapter illustrates how the developed GUI helps in more detailed analysis of specific parameter configurations.

A. CHOOSING OPTIMIZATION PARAMETERS

From the vignette and requirements, the importance of saving fuel is evident when evading from an attacker spacecraft. Assuming that the GN&C subsystem embeds the tactics logic developed in [12] and is responsible for automatically evading from ASAT approaches, one crucial parameter is how frequently the SoI updates its position based on the attacker's position. More frequent updates means undesired fuel consumption. The time between thrust events represent this update rate in the model. So, as a default, SoI activates its propulsion every 540 seconds while the ASAT does the same every 90 seconds. The analysis intends to verify that this updating rate is enough for evasion no matter the direction the attacker approaches from for a four-hour engagement. Design modifications and new requirements can derive from the analysis's results to help the design team improve the system to increase successful evasions.

B. BATCH-RUN EXAMPLE

This batch-run example evaluates the mission's success or failure based on the fulfillment of both requirements, assessing a combination of 26 different attack vectors and four initial distances between SoI and ASAT (Figure 33). Every other parameter in the mission keeps its default value. This first triage aims to identify the attack vectors

concerning the adopted tactics and possible flaws in the automatic maneuvering defense system. This step uses the instance table described in Chapter IV (Figure 32) to assess the 104 different initial configurations (104 rows in the instance table). The SoI could not maneuver effectively to avoid the ASAT approximation in 24 situations. All failures occurred in the same plane, referred to from here as the critical plane. The instance table displays the result of each engagement. However, since the data in the instance table is exportable as a *.xlsx* file, it is simple to display the results using diagrams. Figure 34 summarizes the results and highlights the critical plane to facilitate the visualization of the data in the table.

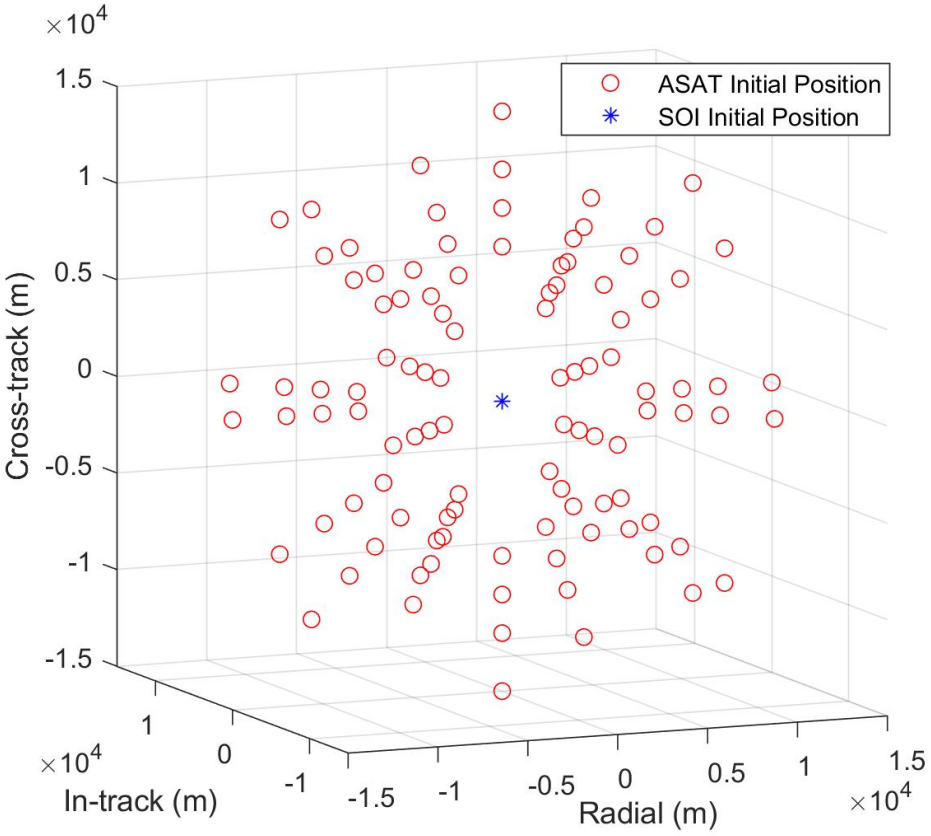

Figure 33. ASAT initial positions (104 cases in total).

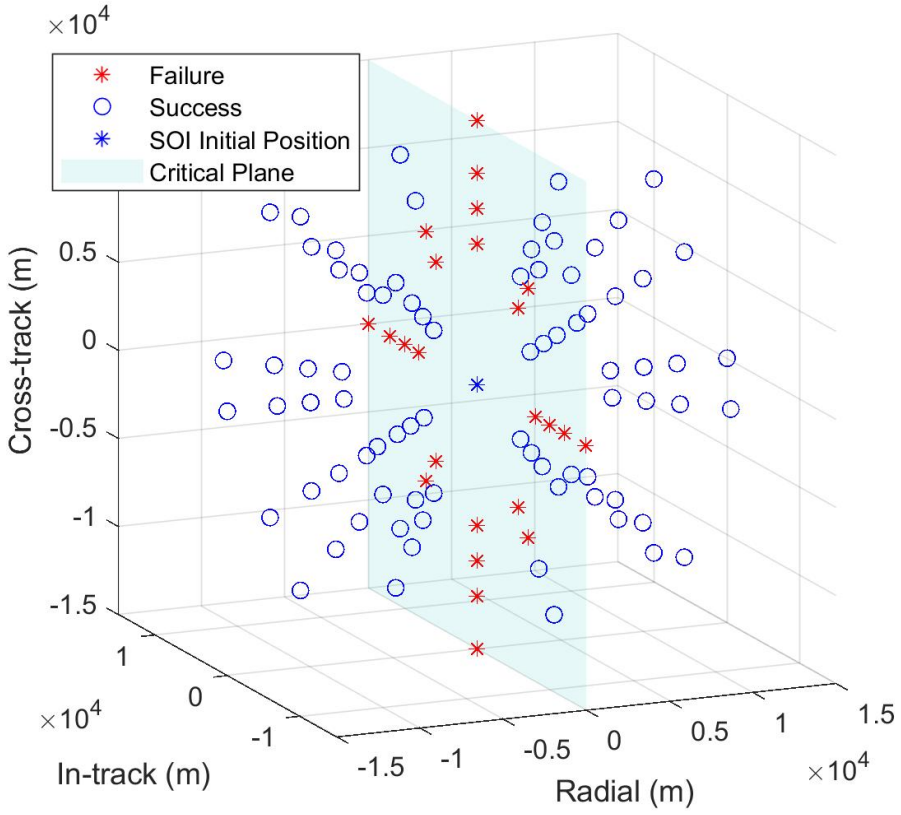

Figure 34. Critical plane concentrates all the failure cases.

C. CRITICAL PLANE ANALYSIS

Figure 35 zooms in the critical plane. This plane is where the radial coordinate is equal to zero. All the attack vectors show failures in this plane. This result reduces the trade space analysis to this plane. This section further studies these attack directions, verifying whether the results remain the same when the SoI updates its position faster than the default value used in the batch run.

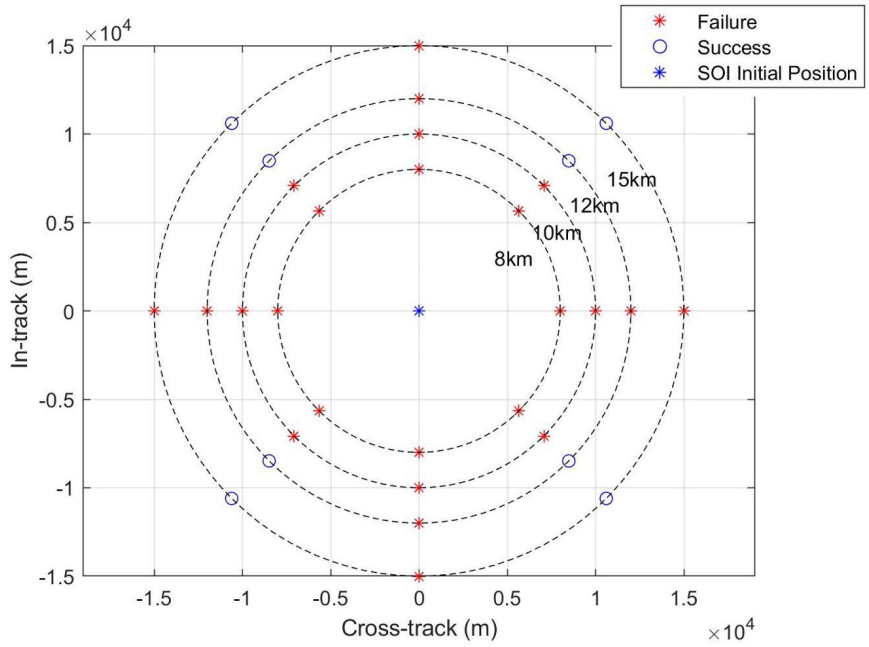

Figure 35. Critical plane zoom-in displaying failure and success evasions.

This time, the simulation considers only the ASAT 32 initial positions in the critical plane. Another 128 rows represent these initial conditions for four different time values between thrust events: 270, 180, 90, and 30 seconds. For updates every 270 and 180 seconds, there is no improvement in the SoI performance, and the results are the same as with updates every 540 seconds. When the update rate is equal to the ASAT (90 seconds), the attack vectors parallel to the in-track axis are not a problem anymore. Yet, the other cases still fail. For 30 seconds between thrust events, there is no improvement from the previous results. In fact, there is a decrease in successful evasions, because the mission starts to fail due to not complying with the fuel requirement in two cases. Figure 36 displays the results.

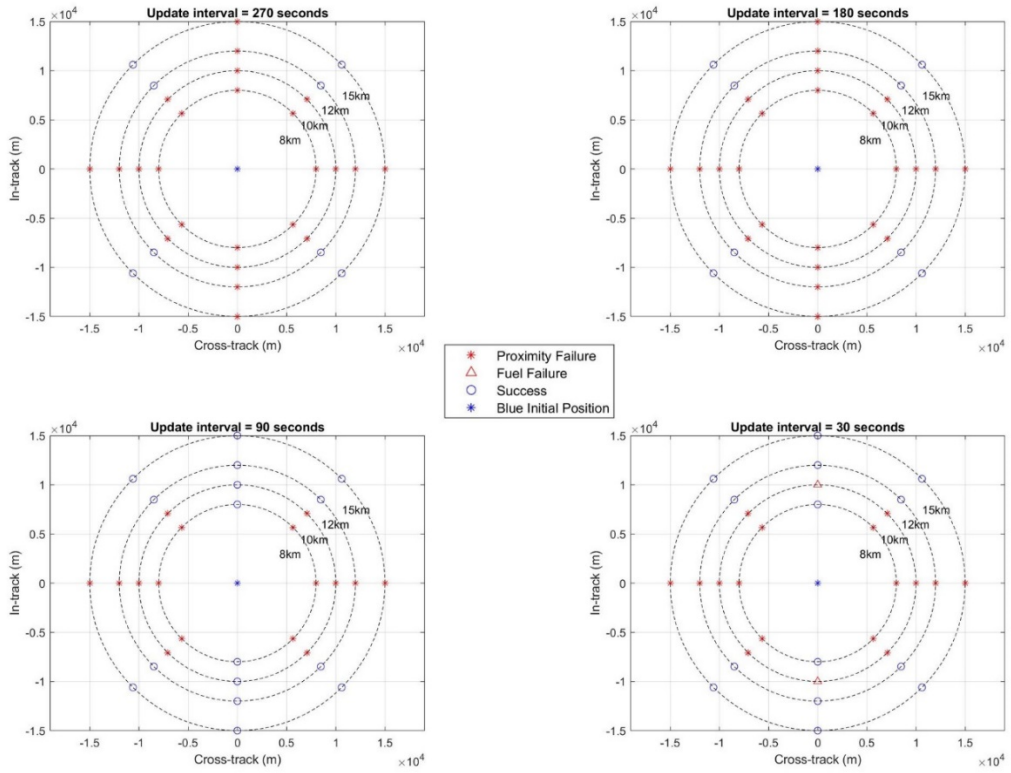

Figure 36. Effect of the thrust event frequency.

From the previous analysis, the best parameter value is 90 seconds. However, there are still six concerning attack vectors. Four vectors present failures within 12 kilometers between SoI and ASAT, which means that evasive maneuvers should start before this limit. On the other hand, SoI failed to prevent the ASAT approximation in all cases when the attack vectors were parallel to the cross-track axis. For this reason, this specific engagement scenario may need a deeper analysis; this, can be accomplished using the GUI presented in Chapter IV. The next section presents these studies.

D. MORE DETAILED DESIGN ANALYSIS

In the GUI simulation, the ASAT starts 8 kilometers away from the SoI in the cross-track axis. During the four-hour engagement, the SoI's position updating rate is 30 seconds. The minimal distance between them is less than two kilometers for this setting. Figure 37 shows the GUI settings and the trajectory plot.

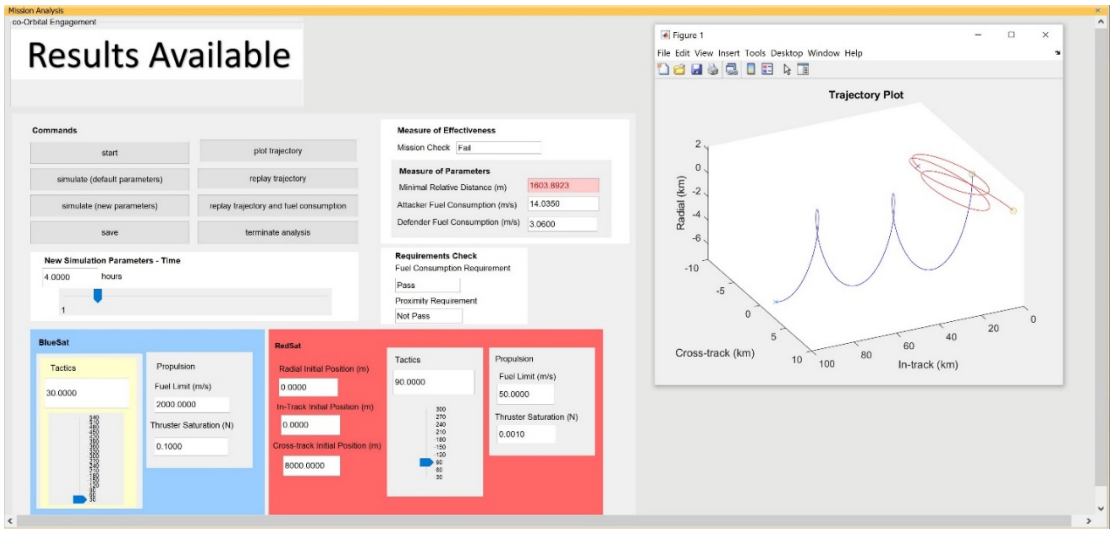

Figure 37. Results and trajectory plot for an engagement with ASAT and SoI 8 kilometers apart in the cross-track axis and 30 seconds between thrust events.

By displaying an animation to the user, the simulation replay trajectory button clarifies better when the proximity requirement failed. This feature allows mission analysts to verify the seriousness of the failure and when it occurred in this situation. Figure 38 shows the kinematic of the engagement.

The minimum separation between the spacecraft occurs at the beginning of the engagement. It is possible to conclude that the satellites do not remain near each other for an extended period. In fact, after that, the SoI is capable of evading and remains distant from the ASAT. So, one can conclude that, in terms of evasion, this situation is acceptable, and the operational requirement established was very restrictive. Also, it is possible to infer that the SoI maneuver is correct since it tries to keep the ASAT in a different plane than the critical plane. The problem is that this maneuver does not occur quickly enough.

Since fewer than two kilometers is too close in space terms, the GUI is further used to verify one combination that increases the distance between them in the ASAT's initial approach attempt. Updating the SoI position every 10 seconds (considered here as the minimum value possible) and considering that the maneuvers start when the ASAT is 15 kilometers away from it results in a minimum distance of approximately 3.5 kilometers. This new result increase means fewer opportunities for the ASAT to collect intel or attack

the SoI. On the other hand, it still does not fulfill the established proximity requirement. It means that this case requires studies considering in the influence of other parameters.

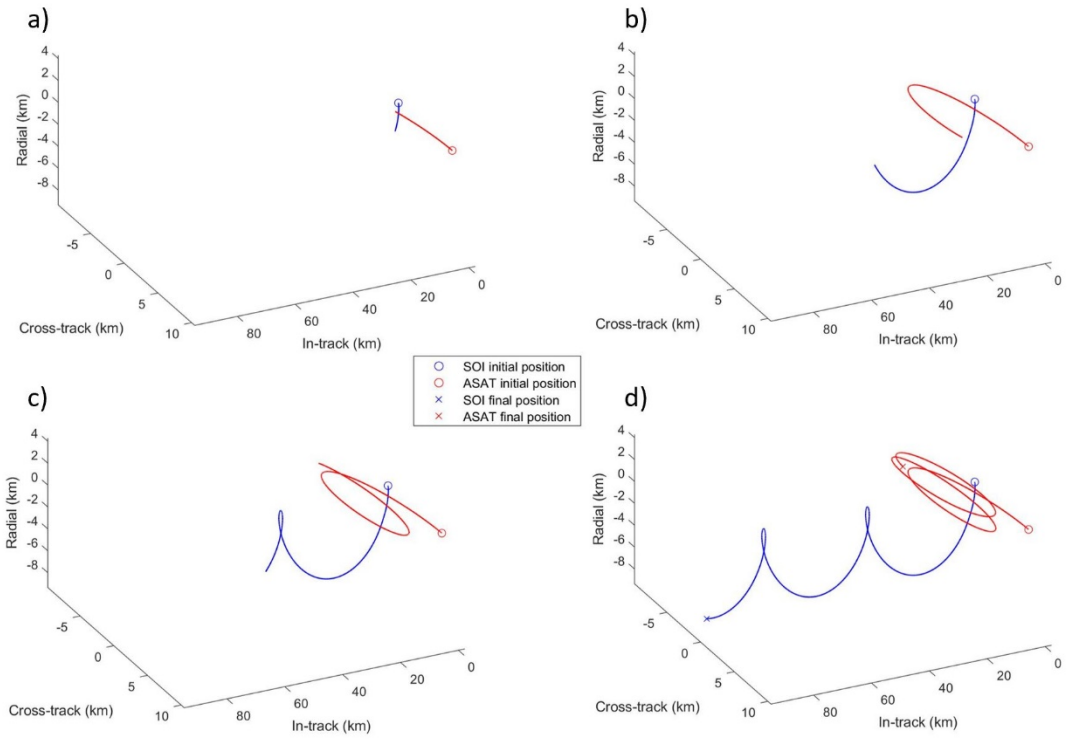

Figure 38. Engagement kinematics: a) Instant where the closest proximity occurs; b) ASAT attempts to engage a second time without success; c) SOI evasion maneuver keeps it safe from ASAT; d) ASAT cannot get closer again in the four-hour period.

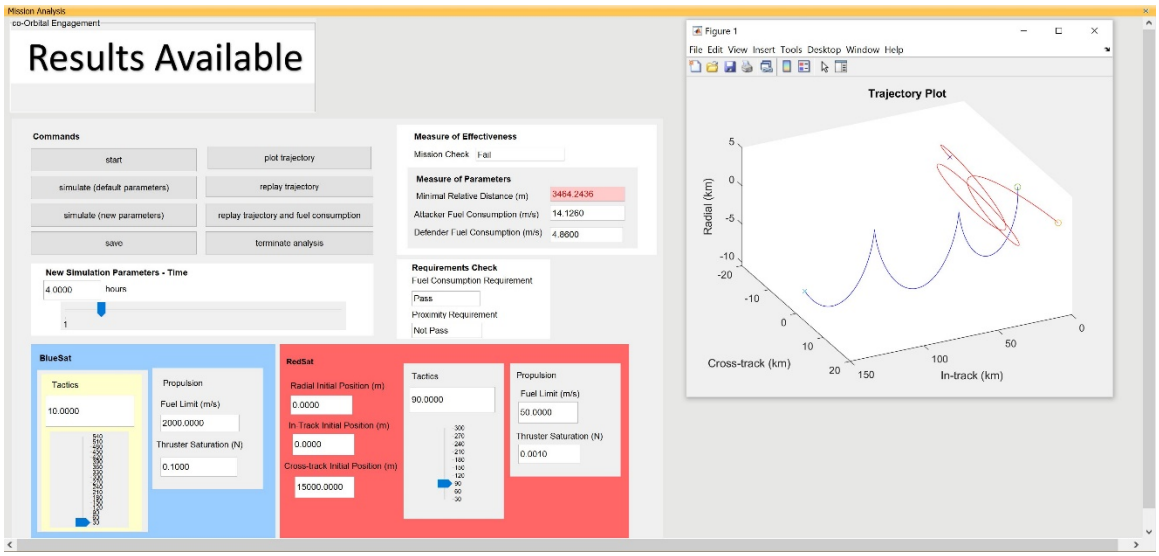

Figure 39. Results and trajectory plot for an engagement with ASAT and SoI 15 kilometers apart in the cross-track axis and 10 seconds between thrust events.

E. EXAMPLE OF OUTCOMES SUPPORTED BY THE DEVELOPED MODEL

Based on the example considered in this chapter, a systems engineer could conclude the following points about the mission design, which are now supported within the developed CSM/Simulink model.

For a specific case of the ASAT being 10 times lighter than the SoI:

- The mission is successful for more than 70% of the engagements simulated using the default parameters and the fuel-saving approach logic for maneuvers.
- There is only one plane that raises concerns in this COE vignette. The critical plane is the one where the radial distance between the SoI and the ASAT is equal to zero.
- Every possible hostile spacecraft must remain out of this critical plane. A possible solution for this is to increase the space situational awareness of the system to activate the tactics block embedded in the GN&C earlier.

This solution requires the elaboration of specific requirements for the sensors in the avionics subsystem.

- Due to its particularity, COEs beginning with the ASAT in the critical plane require specific tactics to increase the radial distance through more effective maneuvers or even spending more fuel in these cases.
- The proximity requirement could be revised to check whether it is not too restrictive. Even with the ASAT getting closer the SoI evaded it successfully.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

This thesis demonstrated how to enhance CSM, a traditional commercially available MBSE software, through its integration with multidomain simulations of systems' dynamics developed in MATLAB/Simulink development environment. It is envisioned that such an integration between system's representations modeled by systems engineers and by domain experts would allow a better understanding of system's behaviors both statically and dynamically and as such enable a more effective and more realistic design process.

To illustrate the proposed approach and give a sense of the resulting executable MBSE model, this thesis used the satellite COE problem and modeled this mission in CSM (using SysML diagrams) as well as in Simulink (using the blocks representing system dynamics). Both models were integrated into an single executable model, which potential in the conceptual design phase of a satellite system was demonstrated in assessing the overall COE mission effectiveness.

The thesis answered the following research questions formulated in Chapter I Section E as follows.

1. Is it possible to employ high-fidelity models integrated with descriptive models to improve the assessment of systems' missions during preliminary design through executable MBSE models?

SysML is very efficient in describing the system's behaviors statically. Nevertheless, some systems such as satellites need a better way to describe their behavior dynamically, even in their early development phases. For that reason, a dynamic simulation is required to assess an emergent system's behavior, verify and validate the preliminary design of the system. The development of executable models aims to fill that gap in the SysML representation.

From the illustration provided in this thesis, it becomes clear that the assessment of a complex system's design and its mission simultaneously is definitely facilitated by the

executable MBSE approach that employs high-fidelity embedded models. In the COE illustration, the developed descriptive model decomposes the SoI and its mission only at a high level, similarly to what happens in the early phases of any system's design. Even though both SoI and COE mission can be further detailed, the developed model is suitable for the integration proposed. Besides, the outputs generated from it were meaningful to improve mission analysis, systems' parameters sensibility analysis, tactics assessment, proposed operational requirements checking, and even new requirements-writing.

2. Can CSM be integrated with MATLAB/Simulink models for the assessment of military satellite missions?

By design, CSM was developed to allow MATLAB and other modeling software to be an external evaluator for mathematical expressions. However, the concern was if this capability is extendable for complex physics-based models such as the Simulink COE model used as an illustration in this thesis and what would it take to fully integrate both tools. It was shown that utilizing the MATLAB workspace such an integration becomes possible. As such, any legacy model developed by the domain expert(s) in the MATLAB/ Simulink development environment (with just a few minor modifications) can effectively be incorporated into and used within the MBSE approach.

3. What are the remaining challenges and current limitations in this kind of integration?

This thesis revealed that the best approach is to use the physics-based models as black boxes nested in the CSM descriptive model to provide more flexibility for modelers. Using this approach, both CSM and Simulink share a common place to write and read data necessary for their respective simulation tasks. Indeed, before the very first run, the model requires the procedures to create a shared workspace visible by both CSM and Simulink (Chapter IV). Unfortunately, at the moment this process cannot be automated because the *matlab.engine.shareEngine* command must be called from the command prompt in MATLAB and the *kill matlab* command from the CST console panel.

Another limitation is that CSM does not support value properties written in the vector or matrix form. Considering that matrix operations and manipulations are one of the

strengths of MATLAB programming, this is considered to be a major limitation. One of possible solutions is the usage of activity diagrams (in CST) containing opaque actions. Unfortunately, this approach requires a good interaction between the engineers responsible for the descriptive model and the ones responsible for the physics-based model, since these diagrams create the interface that maps value properties to Simulink inputs.

The updating of value properties in CSM and variables in the shared workspace also requires some additional efforts. This time, the activity diagrams need to mix the MATLAB commands inside opaque actions with UML structural feature actions. The problem here is that the design of these diagrams is less trivial than for those used to write inputs in the vector and matrix form since the design requires more profound knowledge of UML. This update issue is more evident in the CSM GUI since the user would want to change parameters to perform a proper analysis.

The CSM GUI capability has some room for improvement too. The buttons require a feature that applies logic to lock them when their use is not needed or does not make sense. Another CSM GUI limitation is the lack of integration capability to display graphs and images generated in the MATLAB environment. To overcome that issue in this thesis, some MATLAB coding was necessary to ensure that the figures requested by the user do not pop up in the wrong place.

Lastly, while the current version of CSM allows to conduct parametrical studies resulting in the so-called instance tables, the capability to present these data graphically is absent.

4. What are the benefits of such an approach?

Besides providing a more appropriate dynamic representation of the system's behavior, the integration considered in this thesis also enhances data visualization for each simulation. This improvement is due to the vast MATLAB function library dedicated to plotting graphics and generating animations. More ways to present data means that all kinds of stakeholders can clearly understand the simulation and its results. Graphs also help enable exploration of the design space and the identification of configurations suitable for the established requirements.

Legacy models already developed for systems or mission analysis do not require enormous modifications for integration. Hence, the domain engineers do not have to change their modeling or acquire new skills to develop executable models within a design team. As already mentioned, the only pre-requisite is that the activity diagrams interface with the descriptive and the physics-based models correctly. Since Simulink is widely used for modeling in diverse domains, it has a comprehensive library of customizable blocks that become automatically available to systems' modelers once the integration is set.

To summarize, executable models work as a bridge between subject matter experts and operational, technical, and non-technical stakeholders, and provide more capabilities as compared to just the descriptive models. Even in the preliminary design, these models provide meaningful insights for the design team in a holistic system's perspective without compromising information consistency. Besides, these executable models can evolve during the systems' life cycle, and they present an excellent potential for developing more elaborated mock-ups or even training tools for systems operators. The illustration presented in this thesis proves that the procedures for creating an executable model using CSM and MATLAB/Simulink are the same for representing any system or mission desired.

B. FUTURE WORK

This thesis has presented all the steps necessary to design an executable model using CSM and MATLAB/Simulink models. The developed executable model, however, has only one Simulink model integrated with the descriptive model of the COE mission. Further work could include more Simulink models representing different aspects of the system (e.g., sensors) and their interaction. The interaction can occur so that the data and outputs generated by one model serve as inputs to the another. This new interaction may require some modification in how the data flow was is designed to keep the descriptive model updated. Additionally, this model could evolve to include a high-fidelity model developed in another CST-supported scripting language. This increment would allow the identification of which integration issues and limitations are raised in the executable model and direct how to overcome these issues.

LIST OF REFERENCES

[1] R. Giachetti, "Digital engineering," *The Guide to the Systems Engineering Body of Knowledge* (SEBoK), v. 2.3 R.J. Accessed Feb. 12, 2021. [Online]. Available: https://www.sebokwiki.org/wiki/Digital_Engineering

[2] L. Thomas, "Developing executable system models of complex engineered Systems," presented at the SERC-Navy Research Transition Talks, Feb. 03, 2021. Accessed Feb. 26, 2021. [Online]. Available: https://sercuarc.org/navy-research-transition-talks/

[3] B. Stone, "There is no spoon: U.S. Air Force digital acquisition strategy (Summary)," You Tube, Feb. 07, 2021. [Online]. Available: https://www.youtube.com/watch?v=dEcPlqImjWc&feature=youtu.be

[4] W. Roper, "There is no spoon - The new digital acquisition reality," United States Air Force, Oct. 07, 2020. [Online]. Available: https://software.af.mil/wp-content/uploads/2020/10/There-Is-No-Spoon-Digital-Acquisition-7-Oct-2020-digital-version.pdf

[5] M. Amissah, "A framework for executable systems modeling," Ph.D. dissertation, Dept. of Engineering Management, Old Dominion University, Norfolk, VA, USA, 2018. [Online]. Available: https://digitalcommons.odu.edu/emse_etds/31/

[6] B. Chabibi, A. Anwar, and M. Nassar, "Towards a model integration from SysML to MATLAB/Simulink," *J. Softw.*, vol. 13, no. 12, pp. 630–645, Dec. 2018. [Online]. doi: 10.17706/jsw.13.12.630-645.

[7] Z. Shabbir and A. Sarosh, "Counterspace operations and nascent space powers," *Astropolitics*, vol. 16, no. 2, pp. 119–140, 2018. [Online]. doi: 10.1080/14777622.2018.1486792.

[8] National Air & Space Intelligence Center, "Competing in space," Wright-Patterson AFB, OH, USA, 2018. [Online]. Available: https://media.defense.gov/2019/Jan/16/2002080386/-1/-1/1/190115-F-NV711-0002.PDF

[9] United States Space Command Public Affairs Office, "Russia conducts space-based anti-satellite weapons test," United States Space Command, Jun. 20, 2020. [Online]. Available: https://www.spacecom.mil/News/Article-Display/Article/2285098/russia-conducts-space-based-anti-satellite-weapons-test/

[10] P. Beery and E. Paulo, "Application of Model-Based Systems Engineering concepts to support mission engineering," *Syst. Basel*, vol. 7, no. 3, pp. 44–58, 2019. [Online]. doi: 10.3390/systems7030044.

[11] B. Weeden and K. Pfrang, “Russian co-orbital anti-satellite testing,” Secure World Foundation, August 2020. [Online]. Available: https://swfound.org/media/207051/swf_russian_co-orbital-asat_aug2020.pdf

[12] E. A. Hanlon, “Design strategies and tactics to defeat co-orbital anti-satellite capabilities,” M.S. thesis, Dept. of Syst. Eng., Naval Postgraduate School, Monterey, CA, USA2018. [Online]. Available: https://apps.dtic.mil/sti/pdfs/AD1059897.pdf

[13] U.S.-China Economic and Security Review Commission (USCC), “2015 report to Congress of the U.S.-China Economic and Security Review Comission,” Washington, DC, USA, 2015. [Online]. Available: https://www.uscc.gov/sites/default/files/annual_reports/2015%20Annual%20Report%20to%20Congress.PDF

[14] MilsatMagazine, “Space threat assessment 2020.” Accessed Oct. 21, 2020. [Online]. Available: http://www.milsatmagazine.com/story.php?number=1753711599

[15] United Nations Institute for Disarmament Research, “Counterspace capabilities,” UNIDIR, Geneva, Switzerland, 2018. [Online]. Available: https://www.unidir.org/files/medias/pdfs/counterspace-capabilities-backgrounder-eng-0-771.pdf

[16] Secure World Foundation, “Global counterspace capabilities highlights.” Accessed Apr. 10, 2021. [Online]. Available: https://swfound.org/counterspace/

[17] B. Weeden and V. Samson, “Global counterspace capabilities: An open source assessment,” Secure World Foundation, 2021. [Online]. Available: https://swfound.org/media/207162/swf_global_counterspace_capabilities_2021.pdf

[18] BBC, “Russia ‘tried to spy on France in space’ - French minister,” Sep. 2018. Accessed Oct. 19, 2020. [Online]. Available: https://www.bbc.com/news/world-europe-45448261

[19] B. Weeden and K. Pfrang, “U.S. co-orbital anti-satellite testing,” Secure World Foundation. Accessed Oct. 24, 2020. [Online]. Available: https://swfound.org/media/207055/swf_us_co-orbital-asat_aug2020.pdf

[20] P. Rincon, “Hayabusa-2: Spacecraft’s ‘bomb’ crater found,” BBC, Apr. 2019. [Online]. Available: https://www.bbc.com/news/science-environment-48065282

[21] National Aeronautics and Space Administration, “In depth | Hayabusa 2.” Accessed Oct. 29, 2020. [Online]. Available: https://solarsystem.nasa.gov/missions/hayabusa-2/in-depth

[22] Office of the Deputy Assistant Secretary of Defense for Systems Engineering, "Digital engineering strategy," Department of Defense, Washington, DC, USA, 2018. [Online]. Available: https://fas.org/man/eprint/digeng-2018.pdf

[23] L. Delligatti, *SysML Distilled: A Brief Guide to the Systems Modeling Language*. Upper Saddle River, NJ, USA: Addison-Wesley, 2014.

[24] Office of the Under Secretary of Defense for Research and Engineering, "Mission engineering guide." Washington, DC, USA, 2020. [Online]. Available: https://ac.cto.mil/wp-content/uploads/2020/12/MEG-v40_20201130_shm.pdf

[25] European Space Agency, "Applying MBSE to a space mission – The Clean Space blog," August 28, 2017. [Online]. Available: https://blogs.esa.int/cleanspace/2017/08/28/applying-mbse-to-a-space-mission/

[26] S. C. Spangelo *et al.*, "Applying Model Based Systems Engineering (MBSE) to a standard CubeSat," presented at the 2012 IEEE Aerospace Conference, Big Sky, MT, USA, Mar. 2012. doi: 10.1109/AERO.2012.6187339.

[27] Dassault Systèmes, "Cameo Systems Modeler." Accessed Feb. 12, 2021. [Online]. Available: https://www.nomagic.com/products/cameo-systems-modeler

[28] O. Casse, *SysML in Action with Cameo Systems Modeler*. London, England : ISTE Press, 2017.

[29] Dassault Systèmes, "Cameo Simulation Toolkit." Accessed Feb. 12, 2021. [Online]. Available: https://www.nomagic.com/product-addons/magicdraw-addons/cameo-simulation-toolkit

[30] No Magic Inc., "Integration with external evaluators - Cameo Simulation Toolkit 18.5 - documentation." Accessed Feb. 12, 2021. [Online]. Available: https://docs.nomagic.com/display/CST185/Integration+with+External+Evaluators

[31] D. H. Meadows, *Thinking in systems: A primer*. White River Junction, VT, USA: Chelsea Green Pub., 2008.

[32] Object Management Group, "What is SysML? OMG SysML." Accessed Dec. 15, 2020. [Online]. Available: http://www.omgsysml.org/what-is-sysml.htm

[33] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML - The Systems Modeling Language*, 2nd ed. Waltham, MA, USA: Morgan Kaufmann, 2012.

[34] M. Gagliardi, B. Wood, and T. Morrow, “Introduction to the Mission Thread workshop,” Software Engineering Institute, Carnegie Melon University, Pittsburgh, PA, USA, Rep. CMU/SEI-2013-TR-003, Oct. 2013. [Online]. Available: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2013_005_001_63161.pdf

[35] D. Giadrosich, *Operations Research Analysis in Test and Evaluation*. Washington, DC, USA: American Institute of Aeronautics and Astronautics, 1995. doi: 10.2514/4.862212.

[36] B. S. Blanchard and W. J. Fabrycky, *Systems Engineering and Analysis*, 5th ed. Boston, MA, USA: Prentice Hall, 2011.

[37] S. Friedenthal and C. Oster, *Architecting Spacecraft with SysML: A Model-based Systems Engineering Approach*. Scotts Valley, CA, USA: CreateSpace Independent Publishing Platform, 2017.

[38] No Magic Inc., “MagicDraw architecture made simple.” Accessed: Feb. 17, 2021. [Online]. Available: http://www.nomagic.com/files/manuals/MagicDraw%20UserManual.pdf

[39] E. Hanlon and O. Yakimenko, “Introduction to space dogfighting - What matters in space engagements,” presented at the 2019 IEEE Aerospace Conference, Big Sky, MT, USA, Mar. 2019. doi: 10.1109/AERO.2019.8741819.

[40] “Vector Geometry Tool (VGT) Reference Frames.” STK Help. Accessed May 06, 2021. [Online]. Available: https://help.agi.com/stk/index.htm#stk/referenceframesvgt.htm

[41] No Magic Inc., “Integration with MATLAB - Cameo Simulation Toolkit 18.5 - documentation.” Accessed Apr. 02, 2021. [Online]. Available: https://docs.nomagic.com/display/CST185/Integration+with+MATLAB

[42] No Magic Inc., “Simulink co-simulation - documentation.” Accessed Feb. 16, 2021. [Online]. Available: https://docs.nomagic.com/display/MSI190SP4/Simulink+co-simulation

[43] MBSE Execution, “Simple way to set up interactive and powerful SysML and MATLAB simulation in the shared workspace,” You Tube, Dec. 16, 2020. [Online]. Available: https://www.youtube.com/watch?v=qo9JUhqCuiI&t=294s (accessed Feb. 16, 2021).

[44] MathWorks, “Convert running MATLAB session to shared session - MATLAB matlab.engine.shareEngine.” Accessed Feb. 16, 2021. [Online]. Available: https://www.mathworks.com/help/matlab/ref/matlab.engine.shareengine.html

[45] Ontological Modeling Language, “UML - opaque action.” Accessed Feb. 17, 2021. [Online]. Available: https://opencaesar.github.io/eclipse-vocabularies/build/oml/www.eclipse.org/uml2/5.0.0/UML.html#opaqueaction

[46] K. Kalvit, “Application of an innovative MBSE (SysML-1D) co-simulation in healthcare,” M.S. thesis, Dept. of Mechanical Engineering, Purdue University, Indianapolis, IN, USA, 2018. [Online]. Available: https://core.ac.uk/download/pdf/154759597.pdf

[47] No Magic Inc., “Specifying the language for the expression - Cameo Simulation Toolkit 18.4 - documentation.” Accessed Apr. 13, 2021. [Online]. Available: https://docs.nomagic.com/display/CSTD184/Specifying+the+language+for+the+expression

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California