Evaluating the Efficiency and Scalability of CubeSat Simulations on Low-Capacity Systems: A Comparative Study

Clemente Javier Juan Oliver
Technical University of Munich
TUM School of Engineering and Design
Chair of Pico- and Nanosatellites and Satellite Constellations
Caroline Herschel Straße 100 85521 Ottobrunn - Germany
clemente.juan@tum.de

Abstract

With the exponential growth in CubeSat launches in the recent years, added to the developments in satellite constellations operating jointly, an urge for complex-system implementations and simulations is needed. Yet, the scalability and computational feasibility of simulating CubeSat operations on distributed, low-capacity systems remain under-explored. This study addresses this gap by evaluating the efficiency and scalability of CubeSat simulations across a variety of computing environments, including MATLAB Simulink, C++, and C++ integrated with High-Level Architecture (HLA), both on Windows and Linux operating systems, and also including x86 and ARM CPUs architectures. Through comparative analysis across conventional PCs and a cluster of single-board computers (SBCs), this research critically assesses the viability of leveraging 'low-power,' 'low-cost' computing resources for distributed CubeSat constellations simulations.

Our findings reveal significant challenges in balancing simulation fidelity with computational resource allocation, particularly when scaling up simulations using HLA-based distributed architectures on low-capacity systems. The results suggest that, without substantial enhancements in computational power or accepting compromises in simulation quality, the effectiveness of adopting HLA for CubeSat simulation at scale is limited. These insights challenge prevailing assumptions about the computational efficiency of distributed simulations and underscore the need for innovative approaches to optimize resource use. This study contributes to the broader discourse on satellite simulation technologies, urging a reevaluation of computational strategies to advance the capabilities of CubeSat simulations for accessible space exploration.

1. Introduction

The CubeSat industry has experienced an unprecedented boom, with launches skyrocketing in recent years [1]. This surge not only marks a new chapter in space exploration but also underscores an urgent need for sophisticated simulation tools capable of keeping pace with the rapid development of these compact satellites. As CubeSats evolve into more complex systems, the simulation landscape becomes increasingly diverse [2], spanning from basic single-component setups to intricate multi-system architectures, each with its own set of programming languages and technical challenges. This is particularly relevant in the evolving framework of federated satellite networks, that can be composed by myriads of heterogeneous CubeSats working together [3].

The goal of this work is to present a comprehensive performance comparison of CubeSat simulations. By transitioning from traditional monolithic simulations to a cutting-edge distributed system based on the High-Level Architecture (HLA) standard [4], we explore the computational frontiers of scaling satellite constellations simulations across different platforms, hardware architectures and simulation environments.

Embracing a distributed architecture is a strategy aimed at enhancing system flexibility and responsiveness. Within this architectural framework, subsystems are frequently geographically dispersed, dealing with resource exchange through standardized interfaces like the HLA. The evolution of networking technology, coupled with the growing demands for system flexibility, have established distributed simulations architecture as a dominant solution across various technical systems [5].

Despite the increasing adoption of distributed architectures in these kind of simulations, research on the factors driving the shift and the cost/benefit analysis of transition-

ing from monolithic to distributed architectures have not been that widely studied, such as the case of Crawley et al. [6], where the costs and benefits of transition from one architecture to another are not quantified.

Central to this investigation, the following question arise: how does the implementation of a distributed simulation architecture impact the efficiency and scalability of CubeSat simulations, particularly when constrained by low-capacity, low-cost computing resources? This question not only delves into the technical trade-offs inherent in maintaining high-quality simulations against the limitation of computational resources, but also ventures into the broader implications of these choices for the future trajectory of satellite simulation technologies.

By framing our research within this context, we aim to discuss the pathways through which CubeSat simulation methodologies can evolve, marking a contribution to the ongoing dialogue on advancing space technology simulation frameworks.

2. Background

A real satellite model developed as part of the TUM MOVE-II project is considered for this project. The real CubeSat has been in Sun synchronous Low Earth Orbit (575 km) since late 2018 [7]. The nanosatellite simulator, initially designed in Simulink, comprises several subsystems, including a complete space environment subsystem, thermal management, electrical model, satellite dynamics, orbit mechanics model, etc. Nevertheless, a more complete review of its structure will be given in Section 3.1.

Introduced by Professors Robert Twiggs and Jordi Puig-Suari, CubeSats have revolutionized educational and commercial space exploration by leveraging commercially available components, significantly lowering costs [8]. Classified as S3-SATs (Student, Space, Study Satellite), CubeSats exhibit a cubic shape with dimensions under 10 cm and a weight below 1.33 kg [9]. They serve primarily as educational tools, aiming to teach satellite construction and control, enhance student engineering skills, and facilitate scientific data gathering. Despite their educational focus, CubeSats are also increasingly employed for commercial space experiments.

In order to obtain a more realistic environment for the simulations, when integrating the MOVE-II model into constellation simulations, more specifically into the Federated Satellite Systems (FSS) concept proposed by Golkar et al. [3], a review of the trade-offs related to transitioning this model architecture across the architecture spectrum is studied and presented.

The distinction between monolithic and distributed models here is crucial. A monolithic model consolidates all pertinent information into a single entity, while a distributed model fragments the system into multiple submodels. Each submodel encapsulates a segment of the overall model. The decision on how to partition the model into submodels is a significant design choice, influenced by various factors. For instance, in traffic simulation, the division might be based on geographic regions or traffic types, reflecting the intricacies of the simulated scenario [10]. For the case of this paper, the encapsulated system in the distributed architecture would be represented by the whole nanosatellite model as an entity that communicates with other identical models.

In the scope of providing an adaptable and versatile distributed simulation environment for these FSS, focusing on a Distributed Modeling and Simulation (DM&S) approach, one could think of implementation such as the High Level Architecture (HLA) [4], the Distributed Interactive Simulation (DIS) [11] or their middleware application equivalent: the Data Distribution Service (DDS) [12]. HLA's architecture is specifically tailored for large-scale, interoperable simulation exercises, often with a focus on military applications. In contrast, DIS is traditionally used for real-time military training simulations, emphasizing interoperability and real-time interaction. DDS offers a more generic data distribution framework suitable for a wide range of real-time distributed applications beyond simulations, emphasizing data-centric communication and Quality of Service (QoS) policies.

For the scope of this project an HLA-based architecture will be the chosen one, since it is the most widely used distributed simulation standard as well as the one recommended by NATO [13] for this kind of applications. HLA enables several simulations (federates) to communicate through a runtime infrastructure (RTI), all managed by a Federation Object Model (FOM) that specifies the conditions used to exchange data [4, 14, 15].

3. Methodology

In this study, the CubeSat model acts as the federate, performing an Earth observation mission simulated over a duration of 8 orbits (about 13 hours) with generic observation targets given by a set of vectors including coordinates and time constraints that are tracked. After an initial 5-minute stabilization period, the satellites initiate the tracking maneuvers for each target. Following an initial acquisition phase where the satellite orients itself towards the target (nadir pointing) and stabilizes its position, it initiates a recording period for attitude control error lasting 2 minutes. This process is repeated for almost 80 target tracking maneuvers, taking a total time of 10 minutes each.

With these premises, two main objectives are stablished: firstly, to conduct a simulation performance comparison across various simulation environments and hardware architectures; and secondly, to evaluate the computational

boundaries of distributed simulations within an HLA framework, particularly focusing on the impact of increasing federate numbers. These main points will provide with insights about the challenges of moving from a monolithic to a distributed simulation and will shed light on the behaviour of this type of distributed simulations.

The methodology unfolds from a well-established Simulink model, extensively validated through the MOVE-II project's prolific research output [7].

The Simulink simulation, while functional, may not be the most computationally efficient to scale in conventional PCs. Consequently, a next step includes performing the transition from MATLAB to C++ code. This approach provides with another software application whose components are tightly integrated and designed to work together as a single unit (monolithic architecture as well). This shift not only promises enhanced performance but also facilitates integration into an HLA environment, marking a pivotal step towards achieving a more scalable and distributed architecture.

Initially, when transitioning towards a distributed simulation, a first approach to implement a distributed architecture would be to concatenate several Simulink MOVE-II models without the need of implementing an upper architecture like the HLA. This approach raises critical questions about the resources and rationale behind adopting a complex Runtime Infrastructure (RTI) for distributed CubeSat simulations.

To address these inquiries, three distinct simulation environments are evaluated: MATLAB Simulink, standalone C++, and C++ within an HLA-RTI framework. These environments are tested across two operating systems (Microsoft Windows and Linux) and two hardware architectures (x86-64 and ARM), providing a comprehensive platform for examining the simulation performance under a variety of conditions. The selection of these specific environments and architectures is driven by their relevance to current CubeSat development practices and the need to understand the simulations' performance across commonly used systems. Detailed scenarios for these experiments are further elaborated in Section 3.5.

3.1. Description of the CubeSat Model

The CubeSat Simulink model has a closely interconnected subsystem structure, which allows a comprehensive simulation and yields valuable information that can be extracted from it. In addition to the Simulink blocks, the use of precompiled functions (S-Functions in MATLAB) is implemented, enabling advanced computations with lower computational overhead thanks to their dynamically linked subroutines [16].

Figure 1 depicts the structure of the Simulink model. While the model comprises numerous lower-level subsystems and configurations, this simplified diagram can be used as a reference for the CubeSat's system architecture. The integration of these blocks and functions within the simulation code grants an extensive range of accurate data on the state of the satellite. It is worth noting that all signals can be extracted at any point of the simulation, providing a practical validated model available to use in a wide range of situations.

Moreover, in spite of the model's modular architecture, this model can be described as a monolithic application, due to its tight subsystems' integration, its direct interconnections, and communications and its single codebase [17, 18].

The activity diagram for the model is presented in Figure 2. First, initialization routines are executed following the Mission Control commands and general data from the model is introduced. Afterwards, if no flags were raised, the step function is called, performing a whole loop through the satellite model taking into account the space environment, disturbances and the actual satellite state. Next, with the computed results, a new satellite state is stored and a data logfile is written. Finally, if no error flags raised and all steps were computed, the program executes cleanup and exit functions to terminate the run.

3.2. MATLAB to C++ Adaptation Process

Although the original Simulink model has a variable timestep, in order to adapt it to C++ without major modifications, a fixed timestep of 0.05s was set. This fixed timestep was chosen as it is the smallest timestep used by signal configurations in the Simulink simulation, ensuring that the C++ simulation remains consistent with the original model.

In order to obtain a functional simulation, the Simulink Coder tool was the chosen approach. It proved to be the most reliable and fastest way to achieve a C++ adaptation, although not trivial. A practical C++ version can be obtained when selecting appropriate configuration parameters for the Simulink Coder [19] in order to get wide support for compiling code across different platforms [20, 21], such as the ones depicted in Table 1. Before achieving the final simulation executable, the fact that the initialization code has to be included and some of the MATLAB libraries and S-Functions are not automatically correctly compiled and linked must be taken into account. Moreover, header files need proper tuning as well. Finally, some custom configurations like a log file and performance monitoring tools are included in this step as well.

The transition from MATLAB to C++ preserves all signals, calculated data, and states from the original Simulink model, thus enabling access to a wealth of data for further analysis and integration into the HLA-based satellite fed-

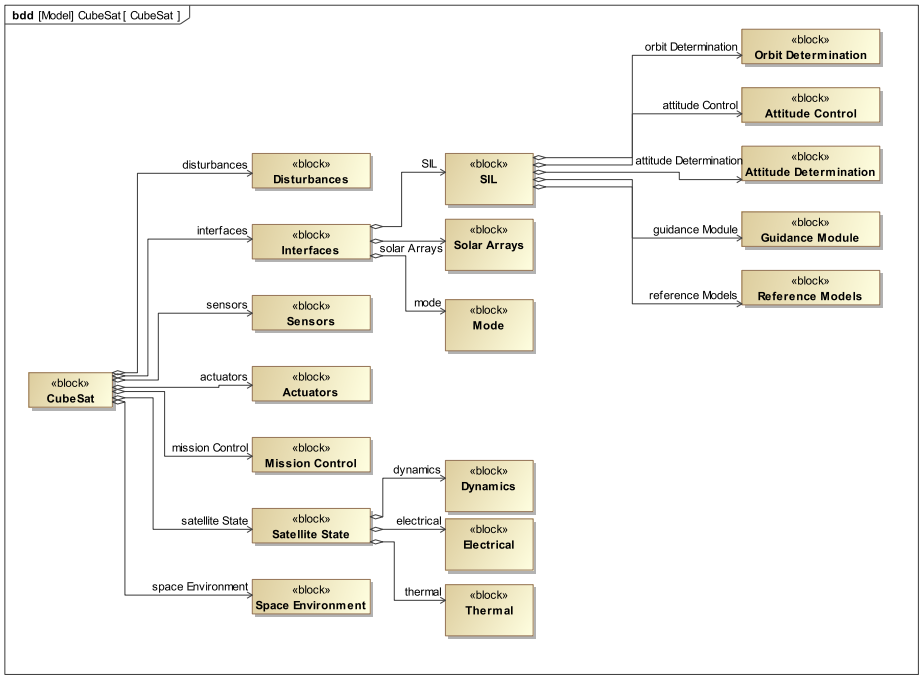

Figure 1. CubeSat model simulator architecture in the SysML modelling language.

Target selection	
System target file	Generic real-time target
Language	C++11(ISO)
Build process	
Toolchain Windows	MinGW64 — CMake/gmake
Toolchain Linux	GNU gcc/g++ — CMake/gmake
Build configuration	Debug

Table 1. Simulink Code main configuration parameters used in this work.

eration simulator. It should be stated that fixed timesteps simplifiy the integration process and reduce potential errors in discrete-event simulations but they also lead to unnecessary computations when system dynamics are slow, affecting performance. Nevertheless, the enhanced performance and flexibility provided brings more benefits than drawbacks despite this fixed timestep. This journey from MATLAB to C++ underscores the significance of strategic decision-making in simulation adaptation, balancing between maintaining model integrity and optimizing computational performance while opening the possibility for advanced data analysis and integration into larger simulation frameworks.

3.3. Integration into the FSS HLA-based Simulator

In order to be able to test satellites in a more realistic environment, working in a federation with more satellites and communicating with systems just as they would in real life, an HLA-RTI system provides with a distributed modelling and simulation environment [22] for the operation of a FSS in a virtual environment. The two main points of the HLA IEEE standard are: promoting interoperability between simulations and aiding the reuse of models in different contexts [23]. Nevertheless, the HLA is an architecture, not a software, therefore the use of RTI middleware software is required to support operations of a federation execution [14]. In this case, the CERTI runtime infrastructure was used. CERTI is an Open Source HLA RTI implementation [24, 25].

CERTI has a characteristic architecture, consisting of two main RTI processes: a local (ambassador) one (RTIA) and a global one (RTIG), as well as linked library (libRTI). This architecture can be seen in Figure 3

More specifically in this case, the CubeSat simulation has to be inserted as a federate, which will locally interact with the Ambassador process (RTIA). Then RTIA and RTIG exchange messages through the network, so that the RTIG can manage the creation and destruction of federation processes and monitor data exchange. RTIG serves as a centralized (intelligent) broadcaster which takes and de-

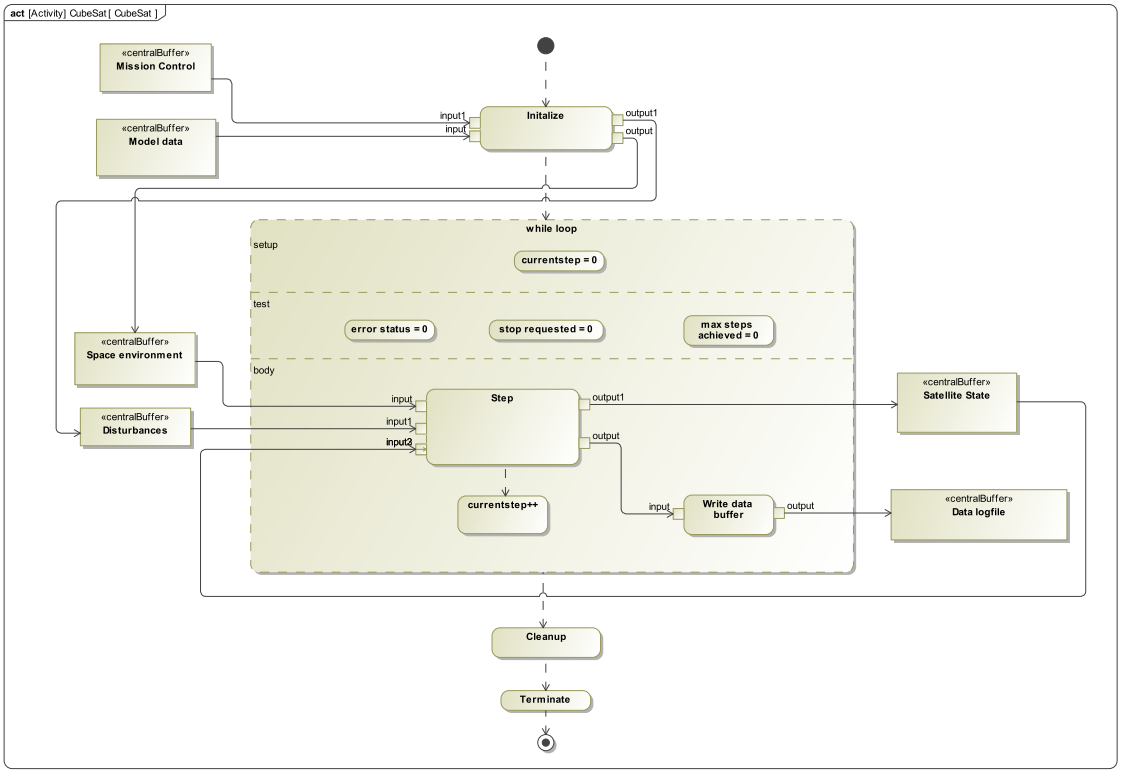

Figure 2. CubeSat model activity diagram in the SysML modelling language.

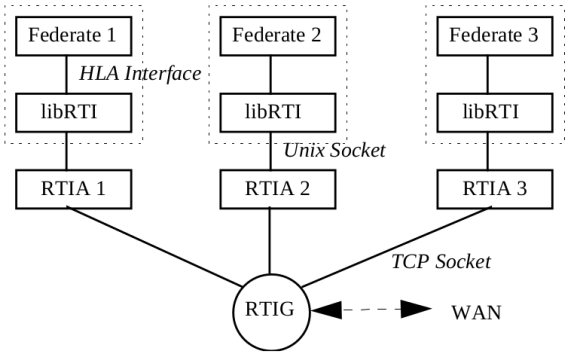

Figure 3. CERTI architecture [24].

livers the messages to the interested RTIAs [26].

With respect to the HLA simulator used, whose architecture is depicted in Figure 4, one can distinguish four types of blocks regarding functionality: Simulation, Control, HLA and Utility.

On a structural point of view, the main file is responsible for including all the necessary libraries (including the CubeSat and other federates ones) and coordinate the different federates attached through the simulator function. Then, the simulator instantiates each model included in it and obtains a simulation object with its attribute values[1] for each one of them. Finally, the HLA function manages each connected model as a federate, creating a unique RTIG and an RTIA for each one of the federates. It assigns one federate to manage the federation, thus allowing the rest to simply join the existing federation. This block also includes the Federation Object Model (FOM), which is a specification defining the information exchanged, including object classes, object class attributes, interaction classes, interaction parameters, Management Object Model (MOM)[2] contents, and other relevant information [4, 14, 15]. As seen in Figure 3, this HLA function is directly associated to the simulator.

In the case of this paper, the orange model block is performed by the CubeSat C++ code. This way, one can add, in principle, an unlimited amount of satellite model federates if the computational power is available. Therefore, with this setup, simulations between different number of federates can be carried out until reaching the computational power limit of the machine, obtaining a function of its performance based on the number of participants taking part in these simulations.

[1] named characteristic of an object class or object instance [4]

[2] MOM provides facilities for access to RTI operating information during federation execution [14]

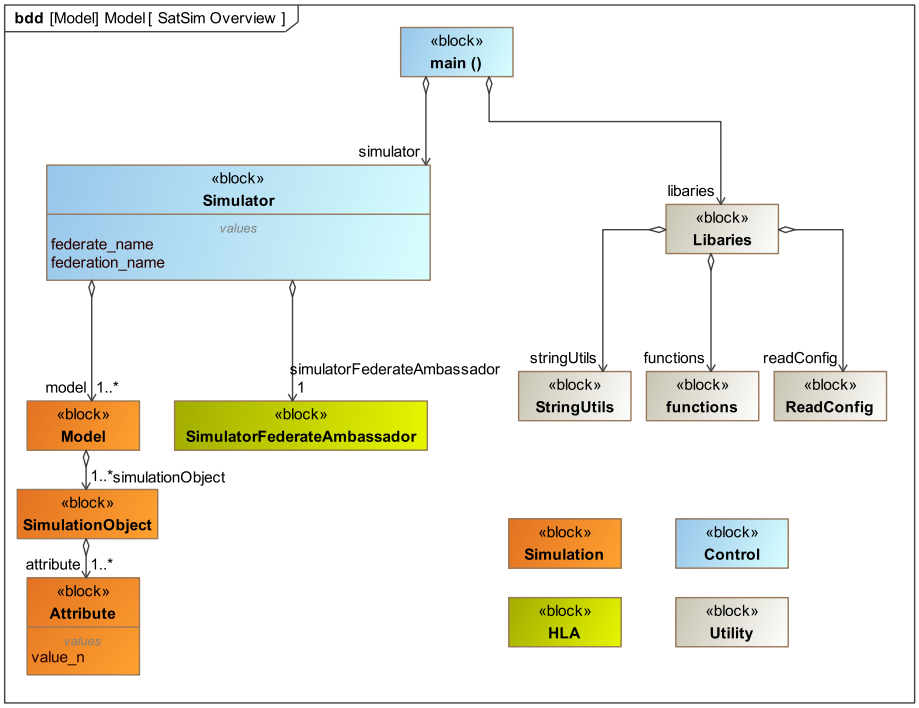

Figure 4. HLA simulator architecture in the SysML modelling language.

3.4. Solver Performance and Data Management Considerations

Finally, some theoretical background and calculations are presented to provide with the complete picture.

In the context of the MATLAB Simulink model, the solver configuration employs the `ode45` solver, an explicit Runge-Kutta method, specifically the Dormand-Prince variant [27]. This choice is pivotal for simulations that demand adaptive timestep control to ensure precision. The `ode45` solver dynamically adjusts timesteps (either contracting or expanding them) to maintain the desired accuracy levels across the simulation. This adaptability, while beneficial for accuracy, introduces variability in computation time, primarily influenced by the model's complexity and the numerical stability requirements [28].

The execution time of the `ode45` solver, used in CubeSat simulations, is influenced by system stiffness, rapid state changes, and precision needs. For stiff equations, `ode45`'s explicit nature may require more steps to find a solution, impacting efficiency. Conversely, it optimizes speed and accuracy for non-stiff problems, despite the higher computational cost associated with its adaptability [29]. The solver's performance is crucially dependent on the dynamics of the system being modeled, including the rate of change in state variables and the complexity of the differential equations [27, 30]. Despite these challenges, `ode45` is favored for its versatility, efficiency in handling non-stiff problems, and integration within MATLAB, making it a suitable choice for CubeSat simulation tasks that demand both precision and adaptability.

Then moving onto the C++ model context, a fixed step (0.05 s) had to be configured in order to generate consistent code with respect to the Simulink model.

In the context of the High Level Architecture (HLA)-based simulation employed within this project, the CubeSat model generates an extensive array of data, including over 3,000 signals and more than 500 states, such as positions, velocities, battery states of charge, etc. Given the substantial volume of data, the potential data transmission rates across the HLA network are significantly high. To manage the complexity and volume of data effectively, the study strategically selects only a limited number of double-precision vector variables for transmission within the HLA framework (position and velocity vectors). Consequently, the size of the data packets broadcasted by the satellite model is quantified at 24 bytes for each attribute value transmitted. This calculation does not account for additional metadata that may accompany the messages, such as federate-specific details.

Moreover, the configuration of the HLA simulator introduces a variable dimension to the simulation's data handling capabilities. The simulator's settings permit adjustments to the message transmission frequency and the synchroniza-

tion strategy among federates. Specifically, it allows for the customization of the operational tempo, including options to synchronize computation steps across all federates before proceeding to subsequent calculations. This adaptability in the HLA simulator's configuration underscores the system's flexibility in balancing data transmission efficiency with computational demands and synchronization requirements within distributed simulation environments [4].

In this study, the messaging protocol is designed to replicate one second of simulation time, taking into account the predefined timestep of 0.05 seconds. Consequently, this protocol stipulates the transmission of a message at every twentieth simulation step. Achieving real-time simulation fidelity necessitates that the cumulative processing of twenty steps and data communication operations—encompassing both outbound and inbound transactions—must be completed within a one-second timeframe. This critical consideration for real-time behavior underscores the importance of optimizing computation and communication workflows to adhere to the simulation's temporal parameters. Future discussions will delve into the temporal efficiency of the simulation, particularly through the analysis of the simulated time per second metric.

In order to determine the computation time, following other related simulations such as the one conducted by Gervais et. al [23], the metric of Worst Case Execution Time (WCET) is the main objective. Since CERTI does not provide any liabilities regarding real-time execution [31], some modifications had to be applied to compute the correct runtime characteristics, such as latency measurements.

3.5. Hardware and Testing Scenarios

Various configurations have been deployed across multiple computers to cover a spectrum of test scenarios. The initial setup involves a laptop that dual-boots Windows 11 and Ubuntu 22.04, powered by an Intel Core i7-8750H processor with 6 cores and 12 logical processors, clocking speed of 2.20 GHz. This system is equipped with 16GB of DRAM, a GeForce GTX 1060 graphics card with 6GB of memory, and relies on HDD for storage. Additionally, there is a group of six PCs, each configured to dual-boot Windows 11 and Debian 12. These computers are outfitted with Intel Xeon E-2146G processors at 3.50GHz, featuring 6 cores and 12 logical processors, 16GB of RAM, NVIDIA Quadro P620 graphics cards with 2GB of memory, and utilize SSDs for storage. The testing array is completed with a cluster of ARM-based single board computers (SBCs), comprising a mix of Raspberry Pi model 4Bs, Raspberry Pi model 5s, and OKdo ROCK 4C+ SBCs. The laptop and SBCs are located in the same building, while the rest of the PCs are located in a different building a few hundred meters apart, although on the same network.

Regarding different environments and platforms, the following cases can be presented:

1. **Simulink Model (Windows)**: evaluation of the Simulink CubeSat model within the MATLAB framework on two distinct Windows-based systems to assess model behavior and performance.
2. **Binary Executable File (Windows)**: deployment of a C++ adapted CubeSat model as an executable within Windows environments to analyze functionality and performance, offering insights into Windows compatibility. This case is also analysed on the two Windows-based computers.
3. **Binary Executable File (Linux x86-64)**: on both Linux distributions, a program containing the same adapted C++ CubeSat model is run to examine its functionality and execution characteristics within this environment.
4. **HLA-based simulator (Linux x86-64)**: across the six-computer setup, a detailed assessment of a simulator that employs a distributed architecture is performed, evaluating its operational effectiveness and behavior.
5. **HLA-based simulator (Linux ARM 64-bit)**: testing the HLA framework on ARM-based CPU architectures (SBC cluster), thus validating the HLA implementation's compatibility and performance on this hardware platform.
6. **Integrated HLA Simulation Across Platforms**: This scenario involves the integration of all three computer types (both Linux x86-64 types of PCs, and Linux ARM 64-bit SBCs) within a singular HLA framework. The goal is to evaluate the interoperability, performance, and scalability of the distributed architecture across heterogeneous computing environments, showcasing the potential for wide-ranging applications.

By conducting these experiments across different operating systems, locations and various scenarios, the aim is to gain a thorough understanding of the system's behavior, performance, and compatibility in diverse computing environments. To sum up, Table 2 summarizes the different setups and simulations that are carried out in each case.

4. Results

4.1. Runtime Analysis

This section evaluates the runtime performance of the CubeSat simulation model across various computational environments and architectures, with a focus on monolithic

Simulation \ Environment	i7 Windows	Xeon Windows	i7 Linux	Xeon Linux	ARM Linux
Simulink Model (monolithic)	X	X			
Binary Executable (C++) (monolithic)	X	X	X	X	
HLA-based Simulator (distributed)			X	X	X

Table 2. Environments and scenarios for the simulations.

and distributed simulations as outlined in Section 3.5. The aim is to shed light on the advantages and limitations of each environment in relation to simulation runtime, thereby guiding optimal environment selection based on specific project requirements.

To facilitate a fair comparison of simulation performance across diverse hardware configurations, we normalize the execution time with respect to the computational resources of each system, specifically the number of processor cores and their clock speeds. This normalization accounts for variations in hardware capabilities, thereby enabling a more equitable evaluation of simulation efficiency. The normalized execution time (NET) is calculated using Equation 1:

$$\text{NET} = \frac{\text{Execution Time (minutes)}}{\text{Number of Cores} \times \text{Clock Speed (GHz)}} \quad (1)$$

This formula adjusts the raw execution time by the total processing capability available to the simulation, providing a metric of minutes per core GHz. Lower values of NET indicate higher computational efficiency, considering the amount of processing power employed.

A comprehensive discussion of these results and their implications will be presented in the following 'Results and Discussion' section (Section 5).

4.1.1 Monolithic Simulation

The MATLAB and C++ monolithic simulations reveal significant differences in execution times, particularly when comparing the MATLAB environment to its C++ counterpart. Table 3 illustrates these differences for a single satellite simulation across Intel i7 and Intel Xeon processors. Notably, the MATLAB model's execution time was approximately ten times slower than that of the C++ model, thus excluding its use for federated simulations due to inefficiency.

Figure 5 presents results comparing normalized execution times for the C++ monolithic simulation for both the Intel i7 and the Intel Xeon, Windows and Linux OS. Tendency lines are also displayed, but results are only showed until reaching the computational limit of each machine. The substantial difference in execution times between MATLAB and C++ simulations suggests inherent efficiencies in C++ for computational tasks. Moreover, the difference between Windows and Linux execution times need further addressing as well.

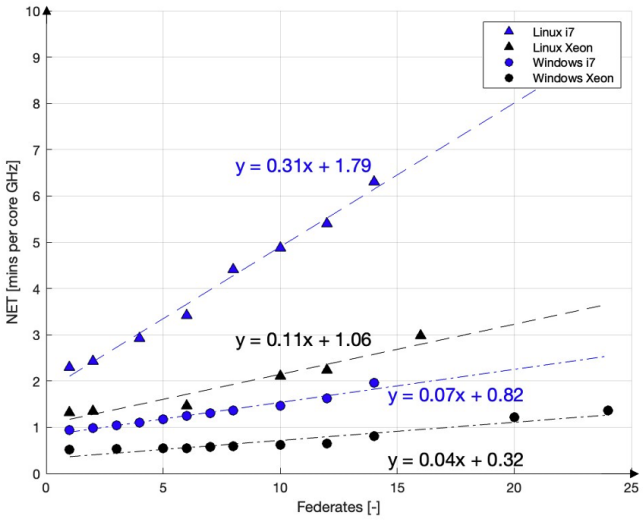

Figure 5. Normalized execution times for C++ monolithic applications in Windows and Linux.

Another important metric that will be evaluated is the simulated time per second (STPS), which quantifies the simulation's computational efficiency. STPS, representing the amount of simulated time (in seconds) that can be processed in one real second, serves as a critical metric for assessing the simulation's capability to run in near real-time or faster. The linear equations provided for STPS across different platforms and processors offer insights into how

Environment / Simulation	i7 Windows	Xeon Windows	i7 Linux	Xeon Linux
MATLAB Simulink (monolithic)	8.11	4.63		
C++ (monolithic)	0.94	0.52	2.30	1.32

Table 3. Normalized execution times for single satellite monolithic simulation on different environments and processors.

the number of federates impacts simulation performance, highlighting the scalability challenges in monolithic environments. In the case of monolithic simulations, the linearised equations are:

- Linux i7:
  $STPS = -1.3067 \times Federates + 26.1347$,
  $R^2 = 0.9584$
- Linux Xeon:
  $STPS = -1.1101 \times Federates + 30.1683$,
  $R^2 = 0.9708$
- Windows i7:
  $STPS = -2.4153 \times Federates + 64.1051$,
  $R^2 = 0.9853$
- Windows Xeon:
  $STPS = -2.1837 \times Federates + 79.4024$,
  $R^2 = 0.9582$

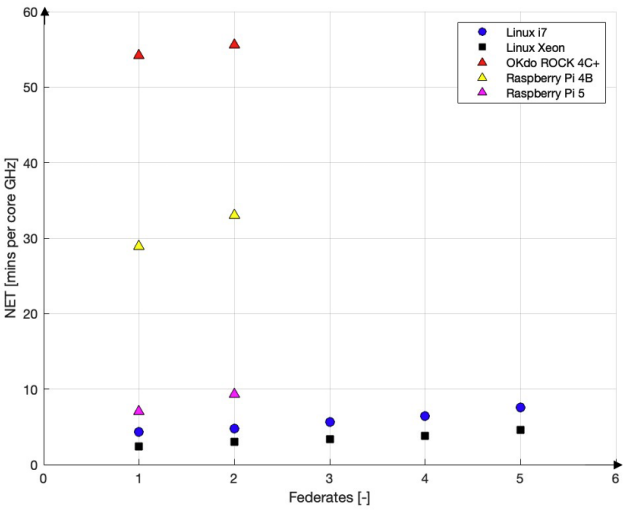

Figure 6. Normalized execution times for the distributed HLA based applications in single machines.

4.1.2 Distributed Simulation

NET for the C++ HLA-based distributed simulations for each type of machine used are presented in Figure 6. These simulations are performed using only one machine for the computation of federates and in parallel, an additional machine running exclusively the RTIG. The displayed computers perform the satellite simulations and send the data through the RTIAs. Here each type of computer's computational power can be easily compared individually before delving into more complex simulations including different types and numbers of machines.

Unlike the monolithic simulation environment, distributed simulations offer enhanced flexibility and scalability, since it enables the effective handling of larger federations by distributing the computational load across multiple machines. This approach significantly impacts the CubeSat simulation's ability to model complex scenarios and interactions within a realistic timeframe. In this case, the linearised equations for single machine HLA simulation execution are not provided, since the distributed approach provides a non-linear scalability behaviour, as presented in Table 4. A comprehensive analysis of this behavior and its implications for CubeSat simulation scalability will be provided in Section 5.

To capture the full scope of distributed simulation performance, Table 4 summarises the amount of satellites simulated with respect to the amount of computers taking part in the distributed simulation. This gives the relationship between the number of machines and the total number of satellites that can be simulated concurrently. Understanding this relationship provides valuable insights into the distributed architecture's capacity to support large-scale CubeSat constellations. Here the ROCK 4C+ boards are not displayed, due to their comparatively lower computing capabilities, ROCK 4C+ boards were primarily tasked with running the RTIG process, a strategic decision aimed at optimizing the overall simulation architecture.

Simulation #	Intel i7 machines	Intel Xeon machines	Raspberry Pi 4B machines	Raspberry Pi 5 machines	Total Federates	NET (mins per core GHz)
1		3			3	1.35
2		3			6	1.35
3		4			4	0.92
4		4			6	1.01
5		4			7	1.07
6		4			8	1.14
7		5			5	0.78
8		5			9	1.01
9				2	4	6.23
10			3		3	11.61
11			4		4	9.39
12			5		5	8.31
13	1	1		1	3	14.21

Table 4. Normalized execution times for various CubeSat simulations across different computational setups.

4.2. Scalability Assessment

Scaling the MATLAB model proved particularly challenging due to its monolithic structure and computational intensity, which significantly impacted performance, therefore only one federate was simulated in this environment. The Simulink model has a tighly interconnected structure, which on top of the computational limitations and difficulties in results' repeatability, makes it technically challenging to add new federates for this environment. In addition to that, it has been proven that encapsulating MATLAB/Simulink programs into HLA-based simulations is not trivial [32].

In contrast, the C++ environment offered more flexibility for scalability, leveraging computing techniques to enhance simulation efficiency. In the C++ environment, multithreading was utilized to manage computational loads more effectively, allowing for concurrent processing of simulation tasks. This technique significantly enhanced the simulation's runtime efficiency. In both Linux and Windows, a *.txt* file is generated with attributes and data to simulate the Workspace data logging of the MATLAB computations, but no communication between federates is carried out in the network.

Finally, the fact that CERTI was implemented only in Linux for the HLA-based simulation should be mentioned. This RTI allows real data exchange across the network. Here, the simulations' computations can be split in as many machines as available. The integration of multithreading with CERTI's data exchange mechanisms highlighted the importance of optimizing thread usage to maintain high performance.

5. Discussion

After presenting the results, more detailed insights can be obtained. The first is the highly inefficient performance of MATLAB compared to C++. Allowing the solver to compute the CubeSat with a variable timestep is not enough for it to come close to the execution efficiency of C++ [33]. The execution discrepancies observed with `ode45` in practical applications can be attributed to its mechanism of timestep adjustment, which, while ensuring accuracy, reacts sensitively to the system's response characteristics.

Moreover, falling below the 1 s/s threshold in simulated time per second (STPS) signals a departure from real-time simulation capability. This threshold is crucial for simulations intended to mirror or predict real-world behavior closely. Below this rate, the simulation's utility for real-time decision-making or live scenario testing becomes compromised, underscoring the importance of optimizing computational strategies.

Regarding communication in the RTI, as previously stated, the use of CPU here has another important processes apart from the main simulation: RTIG and RTIA, which introduce additional computational overhead. The amount of data transferred in total (position and velocity vectors) is approximately 150 MB per federate (RTIA). RTIG adds an extra 50 MB overhead per simulation in order to stablish reliable and secure communication between machines. Particularly in SBCs, where resources are limited, the substantial data transfer volume significantly impacts simulation scalability.

The latency values for the HLA-RTI are in the order of milliseconds, while the messages were sent every second. A deterministic behaviour was also shown, producing consistent results in almost identical run-times (only tenths of milliseconds apart). Therefore one can conclude that these values can be considered as real-time [34, 35].

5.1. Performance Comparison

As depicted in Figure 5, one can clearly state that the best performance in terms of computation efficiency is achieved

in the monolithic Windows C++ environment. It is worth noting that its performance was almost two times more efficient than the following case, monolithic Linux C++. The unexpected performance advantage of Windows C++ over Linux could be partially explained by in-depth system profiling, revealing differences in how each OS handles thread scheduling, memory management, and system calls. These underlying mechanisms, coupled with the specific configurations of our simulation software, contribute to the observed efficiency gap [36].

Regarding the MATLAB environment, it was concluded that it is not worth scaling the system to adapt it to a federation of several satellites, due to its poor execution times compared to C++, the difficulties to implement an HLA environment in MATLAB and its rigid monolithic structure.

In the pure C++ environment, the application of multithreading enabled a substantial reduction in execution times compared to the MATLAB simulations. This utilization of concurrent processing showcases the inherent efficiencies in C++ for computational tasks.

Finally, the HLA environment should only be compared to the Linux one, since they perform the same simulation, with the difference that one has a higher architecture (Figure 3) that uses the very same code as an add-on (federate). In principle, it is clear that the pure C++ version is much more efficient, as can be confirmed comparing the results in Figures 5 and 6. However, taking a closer look at Table .4, and taking for instance the Intel Xeon machines simulations and comparing them to the Linux Xeon results in Figure 5, one can see that the NET values are not much greater than 1 minute per core GHz, while the only results that achieve similar results in the monolithic version are one or two federates, maybe 6, but even those are over 1 minute per core GHz, never obtaining lower results.

Therefore, it can be deduced that the HLA implementation can provide with better results in NET, however, the amount of federates that can take part in the simulations do not show the same scalability performance, not achieving more than 9 federates for a single simulation.

This difference can be explained by the HLA management carried by RTIG, which uses up to 20% of a processor resources (even slightly higher values at peak points), while RTIA consumes around 10% of a CPU computation effort. Although these values vary for different cases, for instance in the case of fewer federates, RTIA can go as low as 7-8% only, while for the higher number of federates these values can achieve 13-14%. All in all, this RTI overhead has a crucial impact on the CPU limit of the machines, specially in the SBCs.

Moreover, the latency was under 3 ms for most of the messages, reaching even less than 0.5 ms in the best cases.

Finally, regarding differences in i7 and Xeon, while both CPUs offer similar core and thread counts, their design philosophies cater to different segments. The Xeon E-2146G is geared towards professional and server environments requiring reliability and performance, whereas the Core i7-8750H is aimed at high-performance consumer laptops prioritizing efficiency and portability. The Xeon E-2146G operates at a higher base and turbo frequency, which can lead to better performance for single-threaded applications like these ones that benefit from higher clock speeds. Furthermore, with a larger cache (12 MB vs. 9 MB), the Xeon E-2146G can store more data close to the processor, potentially improving performance for tasks that require frequent access to large data sets.

5.2. Insights into Trade-offs and Considerations in each Environment

A characteristic fact about the MATLAB environment, as mentioned in Section 4, is the high deviation between different tests and problems with results repeatibility. While the rest of the simulations' results were in the order of hundreds of milliseconds apart for the worst cases, MATLAB simulations's time execution differences were in the order of hundreds of seconds.

One should not forget that MATLAB performance can greatly vary when using different compilers provided in the program. C++ can outperform vectorized MATLAB code by a substantial margin, which leads to important performance gains [36]. There are two main factors contributing to this:

- Utilizing external libraries, such as the ones used for the MOVE project in MATLAB, introduces additional processing time [37].
- Matlab dynamically allocates and frees memory, which becomes especially evident in operations like small matrix multiplication. C++, by pre-allocating memory, avoids the creation of unnecessary temporary matrices, leading to significant performance improvements, more particularly when repeated in loops [38].

However, it is essential to note that C++ code development time is typically much longer.

With all that in mind, despite its lower efficiency, MATLAB boasts mathematically robust built-in routines and toolboxes. The priority is obtaining accurate results, and MATLAB's approach of ensuring correctness before addressing efficiency is key. In contrast, C++ introduces the possibility of subtle errors, like implicit conversions from 'double' to 'int', leading to approximately correct outcomes [36].

Expressing ideas and sharing code with colleagues is also more seamless in MATLAB. The code is not only more

readable and concise compared to C++, but it can also be executed without the need for a compiler.

Furthermore, debugging MATLAB scripts, facilitated by an interactive console and workspace, proves significantly more efficient than debugging in C++. Identifying bugs, such as index calculation errors, is expedited in MATLAB, taking minutes, whereas the same task in C++ might consume hours.

Looking at the monolithic results in Figure 5, the computation limit of the machines is reached at around 16 federates, except in the case of Xeon Windows C++. Making use of the STPS linearised approximations, one can compare these practical limits to the theoretical limit of real-time behaviour. If a constraint of 1s/s is used for the STPS value, theoretical limits for Linux are 19 federates for the i7 processor and 26 for the Xeon. For Windows versions, the limit in the case of the Intel i7 is 26 federates and 35 in the case of the Intel Xeon. All in all, the computational limit is reached before the real-time limit. Nevertheless, one must bear in mind that no real communication exchange between federates is being carried out here.

Now, regarding the HLA-RTI number of federates influence on performance, in theory, the maximum achievable number of federates per machine can be estimated similarly to the previous case, this is, by finding the maximum number of federates a computer can handle without reaching the limit of simulated seconds per second of 1 s/s. If one takes into account further problems in network communication, it might be wise to increase this boundary condition to maybe 2 or 3 s/s as a safety factor in order to avoid any unforeseen situations that may arise. However, HLA provides the flexibility to distribute the computing loads among several machines, thus allowing for a more loose computation load for each computer. Moreover, since the maximum amount of federates that were achieved in a single machine is 5 (only 2 in the case of SBCs), the linear approximation might not provide with accurate results, as was previously stated.

Looking at Table 4, one can see that the total amount of federates taking part is not as big as in the pure C++ case, even though more machines were used. As mentioned, this was mainly due to CPU constraints with RTIG and RTIA processes, which introduced significant computational overhead.

In addition, in these simulations, there was an additional limiting factor: the least powerful machine was the one limiting the communications. If one federate crashed, the whole RTI froze waiting for that one satellite. So although sufficient computational resources were available in number, for these kind of reliable TCP communications, each machine must be able to have enough computing power to withstand all the HLA-RTI overheads.

Finally, with regard to satellite simulations development, based on the project phase, computational power availability, budget constraints, and required simulation fidelity, Table 5 provides guidance on selecting the appropriate simulation environment from the ones studied in this paper. The choice of simulation environment should also consider factors such as ease of setup and use, real-time capability, and scalability for large-scale simulations [39].

6. Conclusions

The HLA-RTI implementation proves to be quite useful regarding scalability and providing a distributed and realistic environment for testing. Thus, its main advantage over the rest of the environments is its flexible, adaptable and functional settings, together with its theoretical "unlimited" potential number of federates. However, it was made clear that in the case of high fidelity simulations which need a great amount of computational power, the maximum number of federates is strictly limited by the computing power of the least powerful machine. One should be cautious when choosing a type of simulation too complex for this kind of environments or budget machines, since HLA overhead played a much more important role than expected, leading to frequent CPU crashes.

Moreover, these advantages come with the respective drawbacks and complications of setting up such a complex system structure, such as adapting the main simulation's code, setting up the FOM, or other crucial federation management files. For this project, the complexity of implementing such an architecture introduced numerous technical challenges that, in hindsight, outweighed the benefits achieved.

It can also be concluded that the HLA environment cannot outperform its pure C++ version if more machines are available to run simultaneously. This could be overcome if a great number of more powerful computers were available, but not in the case of conventional PCs and SBCs.

6.1 Future Research

Another topic to take into account in further investigations is the Simulink Coder configuration: while this paper aimed for debugging optimization, one can also use the efficiency optimization setup, potentially leading to more efficient executions regarding both the Simulink simulation and the C++ adaptations. Nonetheless, the complexity of the adaptation process becomes greater when using this setup [19].

MATLAB also offers an HLA implementation, which can be included in the Cubesat model to represent a more accurate simulation environment to compare with the HLA-based simulator. In this case, the HLA-based C++ simulation could in theory be exactly represented in the MATLAB

Criteria	Recommendation
Project Phase	
	• Early Design (Phases A and B): MATLAB/Simulink recommended for its visual tools and ease of use in conceptual and preliminary design.
	• Integration and Testing (Phase D): C++ or HLA-RTI recommended for higher execution efficiency and more realistic testing environments.
Computational Power	
	• Limited (e.g., SBCs, conventional PCs): Pure C++ recommended for its lower computational overhead. HLA-RTI may be too demanding unless only a few federates are involved.
	• Substantial (e.g., high-performance workstations, clusters): HLA-RTI becomes feasible, offering scalable and distributed simulation capabilities.
Budget Constraints	
	• Tight Budget: Consider the pure C++ environment for its efficiency and potentially lower cost due to less demand on hardware.
	• Flexible Budget: HLA-RTI can be considered if the project benefits from distributed simulations and the budget can accommodate the necessary hardware.
Required Fidelity	
	• High Fidelity Required: HLA-RTI offers a distributed environment that can handle complex simulations with high fidelity, especially when computational resources are not a limiting factor.
	• Moderate to Low Fidelity Sufficient: MATLAB/Simulink for early phases and pure C++ for later phases can provide adequate fidelity for many applications without the complexity and overhead of HLA-RTI.

Table 5. Decision criteria for selecting a simulation environment.

environment, leading to better comparisons and more precise conclusions [40]. Although this very same HLA implementation is not trivial to develop, even making use of this Simulink HLA blockset [41].

In the case of the C++ HLA environment, tests could be carried out in more powerful computers, in order to test the hypothesis that a cluster of HLA-RTI machines can outperform their monolithic counterparts. Moreover, a shorter mission or fewer amount of transmitted data might also help.

Finally, by introducing the FSS concept mentioned, different missions and heterogeneous participants can be networked and effectively exchange data and resources between participants in the federation [3]. Therefore, it would be meaningful to add several types of missions and heterogeneous federates, since in this case all federates were the same model following the same mission guidelines. This will allow to see how different satellites interact between them when carrying out different missions that may involve each other [42].

Acknowledgments

I extend my deepest gratitude to the MOVE-II team for generously sharing their Simulink model, which played a pivotal role in this research. Special thanks are also due to the Chair of Pico- and Nanosatellites and Satellite Constellations for providing the essential single-board computers (SBCs) and PCs that facilitated our simulations.

Furthermore, I wish to express my sincere appreciation to my supervisors, Jaspar Sindermann and Ramón García, for their invaluable insights, expertise, and advice throughout the course of this study. Their guidance was instrumental in shaping both the direction and success of this work.

References

[1] Erik Kulu. "Nanosatellite Launch Forecasts - Track Record and Latest Prediction". In *Small Satellite Conference*, USA, 2022.

[2] Pedro Ângelo Vaz De Carvalho, André Ivo, Guilherme Venticinque, Gustavo Vicari Duarte, Matheus Miranda, and Fatima Mattiello-Francisco. "Simplifying Operational Scenario Simulation for CubeSat Mission Analysis Purposes". In *Proceedings of the 11th Latin-American Symposium on Dependable Computing*, LADC '22, page 125–130, New York, NY, USA, 2023. Association for Computing Machinery.

[3] Alessandro Golkar and Ignasi Lluch i Cruz. "The federated satellite systems paradigm: Concept and business case evaluation". *Acta Astronautica*, 111:230–248, 2015.

[4] "IEEE Standard for Modeling and Simulation (MS) High Level Architecture (HLA)– Framework and Rules". *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, pages 1–38, 2010.

[5] Mohsen Mosleh, Kia Dalili, and Babak Heydari. "Distributed or Monolithic? A Computational Architecture Decision Framework". *IEEE Systems Journal*, 12(1):125–136, 2018.

[6] Edward Crawley, Bruce Cameron, and Daniel Selva. *Systems Architecture: Strategy and Product Development for Complex Systems*. Prentice-Hall, Upper Saddle River, NJ, USA, 2015.

[7] Project MOVE II. `https://warr.de/en/projects/move/move-iib/`. Accessed: September 2023.

[8] N. Krishnamurthy. Dynamic modelling of cubesat project move. `https://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-58432`, 2008. Accessed: August 2023.

[9] Emily Normandy. CubeSats - Space Foundation. `https://www.spacefoundation.org/space_technology_hal/cubesats/`, 2022. Accessed: August 2023.

[10] U. Klein, T. Schulze, and S. Strassburger. Traffic simulation based on the High Level Architecture. In *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*, volume 2, pages 1095–1103, 1998.

[11] Peter Ryan, Peter Ross, and Will Oliver. "Distributed Interactive Simulation Revisited: Capabilities of the Revised IEEE Standard". 2018.

[12] Angelo Corsaro and Douglas Schmidt. *"The Data Distribution Service - The Communication Middleware Fabric for Scalable and Extensible Systems-of-Systems"*. 2012.

[13] Nato simulation standards - stanag 4603. `https://nmsg.sto.nato.int/amsp/hla`. Accessed on September 2023.

[14] "IEEE Standard for Modeling and Simulation (MS) High Level Architecture (HLA)– Federate Interface Specification". *IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000)*, pages 1–378, 2010.

[15] "IEEE Standard for Modeling and Simulation (MS) High Level Architecture (HLA)– Object Model Template (OMT) Specification". *IEEE Std 1516.2-2010 (Revision of IEEE Std 1516.2-2000)*, pages 1–110, 2010.

[16] https://mathworks.com/help/simulink/slref/sfunction.html. The MathWorks Inc. S-Function Documentation . Accessed: August, 2023.

[17] Kevin Hoffman. *Beyond the Twelve-Factor App: Exploring the DNA of Highly Scalable, Resilient Cloud Applications*. O'Reilly Media, Inc., April 2016.

[18] Rahul Awati and Ivy Wigmore. Monolithic architecture. https://www.techtarget.com/whatis/definition/monolithic-architecture, 2022. Accessed: October 2023.

[19] The MathWorks Inc. *Simulink Coder User's Guide*, 2023. https://es.mathworks.com/help/pdf_doc/rtw/rtw_ug.pdf. Accessed: July 2023.

[20] The MathWorks Inc. *Simulink Coder Target Languaje Compiler*, 2023. https://es.mathworks.com/help/pdf_doc/rtw/rtw_tlc.pdf. Accessed: July 2023.

[21] GNU make Manual. https://www.gnu.org/software/make/manual/make.html. Accessed: August 2023.

[22] Zhijie Mao, Lin Zhou, and Yingmei Chen. "A Satellite Communication Simulation System Research Based on HLA and MDIS". In *Proceedings of the 4th International Conference on Computer Science and Application Engineering*, CSAE '20, New York, NY, USA, 2020. Association for Computing Machinery.

[23] Clement Gervais, Jean-Baptise Chaudron, Pierre Siron, Regine Leconte, and David Saussie. "Real-Time Distributed Aircraft Simulation through HLA". In *Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT '12, page 251–254, USA, 2012. IEEE Computer Society.

[24] Eric Noulard, Jean-Yves Rousselot, and Pierre Siron. CERTI, an Open Source RTI, why and how. In *Spring Simulation Interoperability Workshop*, pages 1–11, San Diego, US, 2009.

[25] CERTI Project. https://savannah.nongnu.org/projects/certi. Accessed: July 2023.

[26] *CERTI User Guide*. https://www.nongnu.org/certi/certi_doc/User/CERTI_User.pdf. Accessed: July 2023.

[27] The MathWorks Inc. MATLAB ode45 - Solve nonstiff differential equations. https://mathworks.com/help/matlab/ref/ode45.htmlAccessed: September 2023.

[28] Jay Fleming. System of ordinary differential equations - time complexity of initial value problem. https://scicomp.stackexchange.com/questions/29372/system-of-ordinary-differential-equations-time-complexity-of-initial-value-pro. SciComp Stack Exchange Question. Accessed: September 2023.

[29] Nur Adila Faruk Senan. "A brief introduction to using ode45 in MATLAB". https://www.eng.auburn.edu/~tplacek/courses/3600/ode45berkley.pdf. Accessed: September 2023.

[30] Martha L. Abell and James P. Braselton. Chapter 1 - introduction to differential equations. In Martha L. Abell and James P. Braselton, editors, *Introductory Differential Equations (Fifth Edition)*, pages 1–24. Academic Press, fifth edition edition, 2018.

[31] Albert MK Cheng. *Real-time systems: scheduling, analysis, and verification*. John Wiley & Sons, 2003.

[32] Sangha Choi, Wooshik Kim, and Sugjoon Yoon. "Development of an Encapsulator for Interoperability of Non-HLA Based MATLAB Programs (SIMULINK or M-file) to HLA Based Simulations". *International Journal of Machine Learning and Computing*, 10:176–181, 2020.

[33] Tyler Andrews. "Computation Time Comparison Between Matlab and C++ Using Launch Windows". 2012. California Polytechnic State University San Luis Obispo. https://digitalcommons.calpoly.edu/aerosp/78. Accessed: October 2023.

[34] Jürgen Gotschlich, Torsten Gerlach, and Umut Durak. 2simulate: A distributed real-time simulation framework. In Jürgen Scheible, Ingrid Bausch-Gall, and Christina Deatcu, editors, *ASIM Mitteilung 149 / ARGESIM Report 42*. ARGESIM Verlag, February 2014.

[35] Thom McLean. "Repeatability in Real-Time Distributed Simulation Executions". *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 30(4):383–391, 2000.

[36] PhD AP EcE. Performance Tradeoff: When is MATLAB better (slower) than C/C++?, 2023. https://stackoverflow.com/questions/20513071/performance-tradeoff-when-

is-matlab-better-slower-than-c-c. Accessed: October 2023.

[37] Mathworks. Performance of applications that are developed in MATLAB versus C/C++, 2020. https://www.mathworks.com/matlabcentral/answers/704217-performance-of-applications-that-are-developed-in-matlab-versus-c-c. Accessed: September 2023.

[38] Mathworks. Optimize C/C++ Code Performance for Deep Learning, 2023. https://www.mathworks.com/help/coder/ug/optimize-generic-c-cpp-code-performance.html. Accessed: September 2023.

[39] National Aeronautics and Space Administration. *NASA Systems Engineering Handbook*. NASA/SP-6105, 1995.

[40] The MathWorks Inc. MATLAB Simulink HLA Toolbox. https://mathworks.com/products/connections/product_detail/forwardsim-hla-toolbox.html. Accessed: August 2023.

[41] The MathWorks Inc. MATLAB Simulink HLA Blockset. https://mathworks.com/products/connections/product_detail/forwardsim-hla-blockset.html. Accessed: August 2023.

[42] Nasir Saeed, Ahmed Elzanaty, Heba Almorad, Hayssam Dahrouj, Tareq Y. Al-Naffouri, and Mohamed-Slim Alouini. "CubeSat Communications: Recent Advances and Future Challenges". *IEEE Communications Surveys and Tutorials*, 2019.