Ensuring Time Resolution Compatibility for Simulation Executives, Job Scheduling, and Data Exchange

Edwin Z. Crues, Ph.D.
NASA Johnson Space Center
2101 NASA Parkway
Houston, TX 77058
edwin.z.crues@nasa.gov

Daniel E. Dexter
NASA Johnson Space Center
2101 NASA Parkway
Houston, TX 77058
daniel.e.dexter@nasa.gov

Keywords: Simulation, Time, Time Resolution, Distributed Simulation, HLA, SpaceFOM

ABSTRACT: *Time, as a concept, is something everyone experiences. It seems like a simple concept used to measure the temporal progress of our lives, associated events, and accomplishments. Time is also often a key independent variable in the mathematical formulation of physics-based models of dynamical systems; this is particularly true of space systems modeling and simulation. As a result, many simulation systems use time as the basis for their simulation executives. While a simple concept for most daily activities, time representation quickly becomes more complex and nuanced when used as the independent variable in numerical simulations. This paper explores some key factors for time representations in time-based simulations. It discusses the principal time lines that must be defined and the relationships between those time lines. It analyzes the core time representation within the simulation executive and the relationship between underlying discrete time ticks of the executive and mapping into a continuous floating-point representation of time. This leads to a discussion of model job scheduling and the relationship to the native executive time representation. From this, job scheduling relationships and constraints will be derived based on the executive time representation. Finally, this will lead to a discussion of data exchange rates and constraints for time synchronized distributed simulations. The paper concludes with observation and recommendations on managing time resolution compatibility across time synchronized distributed simulations.*

1 Introduction

This paper should be considered as a prequel to another Simulation Interoperability Standards Organization (SISO) Simulation Innovation Workshop (SIW) paper from the authors [1]. That paper deals with timing constraints for time synchronized time managed distributed simulations. After writing and presenting that paper, many questions were raised related to the fundamentals of the time concepts associated with simulation executives and job scheduling as applied to data exchange between HLA-based distributed simulations [2, 3, 4]. In response, the authors decided to document our understanding of the concepts associated with time, time resolution, and job scheduling cycle compatibility in this paper.

The main topic of this paper is to provide simulation developers with a description of the key principals of how time representations can be used to ensure time resolution compatibility for simulation executives, job scheduling, and data exchange. Many of the concepts covered in this paper are defined and presented in other papers. However, we have not found a single resource where these constituent concepts of time are brought together to describe and define the key relationships between time, time resolution, executive time tick size, job scheduling, and data exchange in distributed simulations. As a result, this paper will draw from some key published resources to lay the groundwork for our discussion. Key time concepts and definitions are from the SISO Space Reference Federation Object Model (SpaceFOM) [5, 6]. Key simulation executive and job scheduling concepts and definitions are from the Trick Simulation Development Environment [7]. Important simulation time representation concepts and definitions are from the SISO 2011 paper [8]. Except where noted otherwise, terminology in this paper conforms with the U.S. Department of Defense Modeling and Simulation

*U.S. Government work not protected by U.S. copyright

(M&S) Glossary [9]. In many cases, we will paraphrase concepts and definitions from these works to aid the reader by presenting the important information within the paper. The reader is encouraged to explore the associated references in detail.

We start by defining the mathematical nomenclature used throughout this paper. This is followed with some basic definitions of time and the principal time lines needed for consistent simulation definition and execution. Next is a brief high-level discussion of simulation executives and the concepts of frames and job scheduling. The paper then explores the details of time representations and the consequences of time resolution mismatches. Real-Time considerations are next; followed by discussions on the additional complexities of ensuring time resolution compatibility in distributed simulations. The paper ends with conclusions and a time resolution compatibility check list.

1.1 Nomenclature

The following nomenclature will be used in the ensuing discussion and the formulation of the constraint equations.

dt	Simulation time step.
dt_{cc}	The difference between the current and desired CCT value.
dt_{eo}	Simulation executive frame overhead.
dt_{fm}	Simulation executive real-time frame margin.
dt_{fp}	Processing time for a given executive frame.
dt_{ji}	Simulation scheduled job execution time.
dt_{rt}	Simulation executive real-time frame.
dt_{ω}	The cycle time step associated with a given frequency (ω).
dt_{τ}	The time step corresponding to 1 Tick. The minimum time quanta.
D_{ω}	Denominator for fractional representation of job cycle time.
ms	Millisecond ($10^{-3}s$).
M_e	General simulation executive time tick Multiplier (Ticks/sec).
M_f	Federate executive time tick Multiplier (Ticks/sec).
M_{hlt}	Distributed simulation (HLA) logical time tick Multiplier.
M_p	The minimum prime factor executive multiplier for the simulation.
n_f	Number of frames since the start of the simulation.
n_j	Number of scheduled jobs in a given frame.
n_{τ}	Number of executive ticks since the start of the simulation.
N_{ω}	Numerator for fractional representation of job cycle time.
R_f	The federate to federation multiplier ratio (M_f/M_{hlt}).
s	Second.
T	Simulation executive time in units of Ticks (τ).
T_f	Federate executive time in units of Ticks (τ_f).
T_{hlt}	Federation execution time in units of Ticks (τ_{hlt}).
t_{cc}	Computer clock time.
t^*_{cc}	Desired computer clock time.
t_{cc0}	Computer clock time epoch at start of simulation.
t_{hlt}	Distributed simulation (HLA) time.
t_{se}	Simulation elapsed time.

t_{ss}	Simulation scenario time.
t_{ss0}	Simulation scenario time epoch.
ϵ	The smallest time step available to the executive.
μs	Microsecond ($10^{-6}s$).
τ	Executive Tick.
τ_f	Executive Tick for a specified federate.
τ_{hlt}	Federation HLA Logical Time Tick.

2 Executive Time Concepts and Definitions

A common simulation executive formulation is to provide a cyclic processing loop where a defined set of computational algorithms are executed in each cycle. Each computational cycle is commonly referred to as a frame. This is the fundamental design for the Trick Simulation Development Environment (Trick) [7]. Simulation systems typically associate the execution of these frames with the monotonic advancement of an independent variable meaningfully associated to a problem being investigated. For many physics-based problem sets, this independent variable is time (t). Each executive frame can then be associated with an incremental change in time (dt).

While most modern computers do maintain concepts of time, most computational processes are not directly tied to time but execute computations sequentially as fast as possible. Time passes but the simulation executive's processing is not constrained by time. The rate of computation is a function of the complexity of the computations and the performance of the computer processors. Therefore, the passage of physical time modeled by computer clock time and simulation time are not directly related. In most cases, this is not an issue. People generally, want programs to run as fast as possible. Initial discussions in this paper will focus on the case where the executive is not time constrained. Specifically, the simulation execution time is not coordinated with the computer clock time. Subsequent discussions will address coordination between the physical time and the simulation time lines. The process of coordinating these time lines is often referred to as time management.

2.1 Time Lines

To this point, three concepts of time have been mentioned: physical time, computer clock time, and simulation execution time. Admittedly, this can be confusing. Therefore, it is important to clearly define and understand the time concepts important to simulation. These simulation time concepts can be categorized as time lines. The SpaceFOM defines the following 6 distinct time lines associated with general and distributed simulation execution [5]:

Physical Time (PT): the non-spatial dimension associated with our spacetime continuum in which events are ordered in irreversible succession from the past to the present to the future.

Computer Clock Time (CCT): the representation of time maintained and managed by the computer usually using some form of oscillator counting oscillations from a known epoch and mapped into an approximation of PT.

Simulation Elapsed Time (SET): the time measure associated with an individual simulation starting at zero and advancing monotonically in quantifiable steps.

Simulation Scenario Time (SST): a model within a simulation that associates the Simulation Elapsed Time with a representation of the problem's Physical Time.

HLA Logical Time (HLT): the time line used by HLA to order messages, regulate execution time advance (Time Advance Request (TAR) & Time Advance Grant (TAG)) and enable deterministic behavior in distributed simulation.

Federation Scenario Time (FST): a time associated with the physical systems being modeled in the participating federates in the federation execution.

These time lines are important in understanding how the progression of simulation time is managed within a simulation, coordinated with other distributed simulations, and coordinated with physical time when necessary. The reader is encouraged to explore the time section of the SpaceFOM document for more detailed discussions of time and time lines [5].

2.2 Execution Loops

Having now defined fundamental simulation executive time line concepts, how are times actually represented within the computations of the executive and how are those representations associated with the afore mentioned time lines? An excellent overview of time representations is provided in the SISO SIW 2011 paper titled "Time Representation and Interpretation in Simulation Interoperability – an Overview" [8]. It discusses challenges of discreet time representations and their use in digital simulations. One observation is that an integer representation of executive time is preferred over a purely floating-point representation. This implies that there is an accompanying model to convert and compare between the native integer representation of time in the executive and common decimal or calendar representations of time.

We can start with a simplistic design for a simulation executive as shown in Figure 1. First, this simulation executive is fundamentally a computer program. It has a start and a shutdown (exit) state like almost all computer programs. In addition, the simulation part of the program has at least four modes or phases of operation: initialization, run, freeze, and shutdown [7].

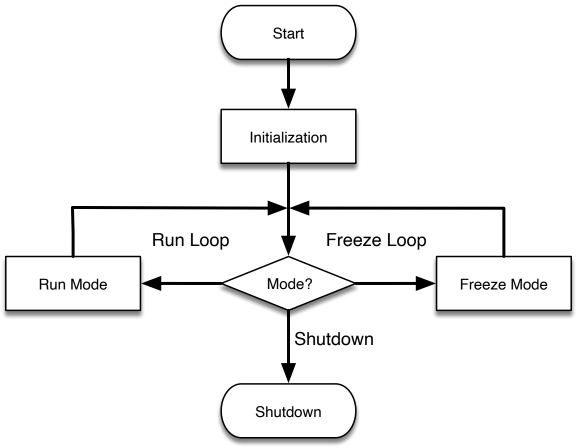

Figure 1: **A simple simulation executive.**

Initialization: The entry mode for a simulation where methods are called to initialize the state of the simulation executive and all the constituent models. Time lines are typically established and defined here but time is not advancing.

Run: Generally, the principal execution loop in a simulation where model jobs are called and time lines are coordinated and advanced.

Freeze: An alternate execution loop in a simulation where special "freeze" jobs are called. While many of the simulation related time lines are not advanced in this mode, the simulation executive may maintain and manage a freeze loop executive counter.

Shutdown: The exit mode for a simulation where methods are called to close out the simulation executive, safe any related systems, release resources, and terminate the program.

2.3 Frames and Job Scheduling

A principal responsibility of a cyclic executive is to coordinate the order and frequency of the constituent model function calls that compose a simulation. A specific model function called by the executive is referred to as a *job* [7]. A common executive approach is to cyclically schedule job calls by frequency.[1] Jobs are executed sequentially in repeating cycles with different job frequencies interleaved appropriately. Ultimately, job frequencies are translated into executive times in the Simulation Execution Time (SET) time line and the executive calls all jobs scheduled to run on that schedule cycle. The specific cycle that a group of jobs are scheduled to run is often referred to as a *frame*.

Some executives define special frames that are used to define SET time intervals for checking simulation execution state. For instance, Trick defines two important executive frames: Software Frame and Freeze Frame. The Software Frame is used in the run loop to schedule when the executive will perform tasks like check for real-time, external synchronize, and inject data value changes. The Freeze Frame serves a similar function for the freeze loop.

2.4 Time Representations

While time is relevant in all modes of the simulation, **Run** is the principal mode where time is advanced. The Trick executive is representative of many common simulation executives. The executive uses an integer counter to keep track of "Ticks" of the executive. A Tick, by definition, represents the smallest discreet increment possible by the executive. Note that we did not say time. The executive accounts for execution cycles using multiples of a Tick. Ticks are used for controlling the execution cycles and for scheduling jobs within execution cycles.

The typical simulation executive maintains a simple linear "model" of time by defining a multiplier (M_e) that provides a scaling between Ticks and SET:

$$T \equiv n_\tau = t_{se} * M_e \quad \text{or} \quad t_{se} = T/M_e \tag{1}$$

where n_τ are the number of executive Tick cycles since the start of the simulation and the definition of T and M_e is a rational fraction strictly greater than 0 ($M_e > 0$). Note that the smallest time increment available to the executive is ϵ:

$$\epsilon \equiv dt_\tau \equiv 1/M_e \tag{2}$$

Typically, the units of M_e are Ticks per second (τ/s). A commonly chosen M_e is 1,000,000 (10^6) Ticks per second. In this case, the value of ϵ is equivalent to 1 microsecond (μs) ($10^{-6}s$). The executive cannot represent a time below 1 μs. This means that all executive time elements must be representable as integer multiples of 1 μs.

This can be a limiting restriction when dealing with some commonly used cycle time (dt_ω) frame scheduling values. For instance, a 128 Hz cycle frequency has a 0.0078125 second cycle time (dt_{128}). Note that this requires a resolution precision of a minimum of 0.5 μs. However, the typical ϵ value is 1.0 μs which is too large to exactly represent the needed 0.5 μs.

$$\begin{aligned} dt_{128} &= 1/128\ s = 0.0078125\ s \\ &= (1/128) * 1,000,000\ \tau = 7,812.5\ \tau \end{aligned}$$

To accommodate 128 Hz and the 0.5 μs remainder, M_e would need to be at least 2,000,000 Ticks per second.

$$\begin{aligned} dt_{128} &= 1/128\ s = 0.0078125\ s \\ &= (1/128) * 2,000,000\ \tau = 15625\ \tau \end{aligned}$$

More detailed discussions on the trade between maximum and minimum executive time representations can be found in [7], [8], and [10].

Note that some cycle frequencies result in cycle times that cannot be exactly represented by floating-point numbers. For instance, a 30 Hz execution cycle will result in a decimal value that cannot be exactly represented (0.03333...). Fortunately, using the Tick multiplier approach does allow for an exact representation in

[1]Trick schedules jobs using job interval times instead of frequency. However, these interval times are equivalent to the inverse of a cycle frequency. For instance, a 10 ms job runs at 100 Hz frequency.

the executive. An executive cycle time of 30 Hz can be exactly represented with an M_e of 3,000,000 (τ/s). Specifically,

$$\begin{aligned} dt_{30} &= 1/30\ s = 0.0333\ldots\ s \\ &= (1/30) * 3,000,000\ \tau = 100,000\ \tau \end{aligned}$$

Therefore, a cycle frequency of 30 Hz can be exactly represented by an executive count of 100,000 Ticks with an M_e of 3,000,000.

What is necessary to support both 128 Hz and 30 Hz?

Assume that schedule cycle time must be representable as a rational integer fraction, $dt_\omega = N_\omega/D_\omega$. The challenge is to find the smallest common Tick multiplier (M_p) that can be used to represent all the desired cycle times. This is equivalent to finding a fractional representation of the cycle time for all scheduled jobs that share a common denominator (D_ω); specifically, the Least Common Denominator (LCD). Fortunately, we can use the process of prime integer factorization to do this [11, 12].

$$\begin{aligned} 1/128 &= 1/(2^7) = (3*5)/(2^6)*(2*3*5) = 15/1920 \\ 1/30 &= 1/(2*3*5) = (2^6)/(2^6)*(2*3*5) = 64/1920 \end{aligned}$$

Therefore, an $M_p = 1920$ results in the following:

$$\begin{aligned} dt_{128} &= 1/128\ s = 0.0078125\ s = 0.0078125 * 1,920\ \tau = 15\ \tau \\ dt_{30} &= 1/30\ s = 0.0333\ldots\ s = 0.0333\ldots * 1,920\ \tau = 64\ \tau \end{aligned}$$

However, a value of $M_e = 1,920$ (τ/s) may not have the timing resolution desired. The corresponding minimum time resolution is $\epsilon = 5.2083\ldots\ 10^{-4}s$ or a little more than 1/2 ms. It is likely that the executive will require better time resolution. Fortunately, any M_e that is an integer multiple of 1,920 (M_p) can support both 128 Hz and 30 Hz. Choosing $M_e = 1,920,000$ supports both 128 Hz and 30 Hz frequencies and has an $\epsilon = 5.2083\ldots\ 10^{-7}s$, or just over 1/2 μs.

$$\begin{aligned} dt_{128} &= 1/128\ s = 0.0078125\ s = (1/128) * 1,920,000\ \tau = 15,000\ \tau \\ dt_{30} &= 1/30\ s = 0.0333\ldots\ s = (1/30) * 1,920,000\ \tau = 64,000\ \tau \end{aligned}$$

If we need additional frequencies, we follow the same process. If an additional frequency falls within the existing integer factorization, then the corresponding M_p is still valid. For instance, adding a 10 Hz frequency:

$$1/10 = 1/(2*5) = (2^6*3)/(2^6*3)(2*5) = (2^6*3)/(2^7*3*5) = 192/1920$$

$$dt_{10} = 1/10\ s = 0.1\ s = 0.1 * 1,920,000\ \tau = 192,000\ \tau$$

However, adding a 100 Hz frequency will not work with the existing integer factorization of 1920. Note that 100 is not an integer divisor of 1920, 1920/100 = 19.2. To find a common factorization an additional integer factor of 5 is required:

$$\begin{aligned} 1/128 &= 1/(2^7) = (3*5^2)/(2^7*3*5^2) = 75/9600 \\ 1/100 &= 1/(2^2*5^2) = (2^5*3)/(2^7*3*5^2) = 96/9600 \\ 1/30 &= 1/(2*3*5) = (2^6*5)/(2^7*3*5^2) = 320/9600 \\ 1/10 &= 1/(2*5) = (2^6*3*5)/(2^7*3*5^2) = 960/9600 \end{aligned}$$

This means that any M_e that is an integer multiple of 9600 (M_p) will work. Fortunately, 1,920,000 is an integer multiple of 9600. Resulting in the following:

$$\begin{aligned} dt_{128} &= 1/128\ s = 0.0078125\ s = (1/128) * 1,920,000\ \tau = 15,000\ \tau \\ dt_{100} &= 1/100\ s = 0.01\ s = (1/100) * 1,920,000\ \tau = 19,200\ \tau \\ dt_{30} &= 1/30\ s = 0.0333\ldots\ s = (1/30) * 1,920,000\ \tau = 64,000\ \tau \\ dt_{10} &= 1/10\ s = 0.1\ s = (1/10) * 1,920,000\ \tau = 192,000\ \tau \end{aligned}$$

Where $M_e = 1,920,000$.

An executive will use these frequency-to-Tick relationships to group jobs into execution frames as a function of Tick count. The Tick count is directly related to SET by M_e and the number of executive Ticks since the beginning of the simulation (n_τ) (see Equation 1).

2.5 Consequences

What are the consequences of not having an executive Tick multiplier (M_e) that is compatible with job cycle time (frequency) representations? To explore, consider the following very simple example.

Assume a simulation that has one single job scheduled with an executive cycle time of 0.25 s (4 Hz). Now assume that the executive has an executive Tick multiplier of 10 ($M_e = 10$). The job cycle time and the executive Tick multiplier are not compatible. In this example, $\epsilon = 1/10 = 0.1s$ while the job cycle time is 0.25. Note that the simulation executive does not have the resolution to represent the 4 Hz job. Specifically, the executive's resolution of $0.1s$ cannot represent the 10 ms resolution requirement for a 4 Hz job ($dt_4 = 0.25$).

What happens when a job cycle time (frequency) is incompatible with the executive's base resolution is executive dependent. There are too many ways to design a simulation executive to analyze them all here. However, we can explore the consequences of two variations on two specific job scheduling approaches: truncated time based, rounded time based, truncated Tick based, and rounded Tick based.

We will start by looking at the time-based variations of job scheduling based on SET times (t_{se}). When job cycle times are incompatible with the executive Tick time step, the computation that translates a job schedule time into a Tick-based time (T) will not result in a whole executive Tick value; there will often be a non-integer remainder. When this happens, Ticks can be computed by either truncating the value or rounding to the nearest whole integer Tick value:

$$T = \mathbf{trunc}(t_{se} * M_e) \tag{3}$$

or

$$T = \mathbf{round}(t_{se} * M_e) \tag{4}$$

For truncation using Equation 3 the cumulative simulation elapsed time (SET) will be scaled by the executive multiplier (M_e) and then truncated to the whole integer Tick value. For rounding using Equation 4 the cumulative SET will be scaled by the executive multiplier (M_e) and then rounded up or down to the nearest whole integer Tick value. The first three (3) seconds of the 4 Hz example simulation execution are shown below in Figures 2 and 3:

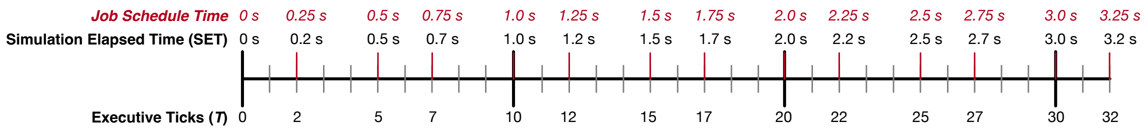

Figure 2: **Example Truncated Time Job Scheduling.**

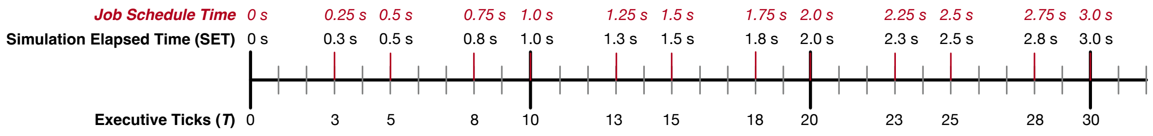

Figure 3: **Example Rounded Time Job Scheduling.**

Note that in neither of these cases can all the jobs align with an executive Tick value (T) or the corresponding SET value (t_{se}). Some job schedule times will match but others will not. Whether truncating or rounding, some job schedule times will not align and are effectively moved forward or backward a Tick. In order for the job schedule times to align with the SET times, the job cycle times must be compatible with the executive time resolution.

Next, we look at the Tick-based variations of job scheduling using the cumulative count of executive Ticks based on the job cycle time (dt_ω). When job cycle times are incompatible with the executive Tick time step, the computation that translates a job schedule cycle time into a Tick-based time (T) will not result in a whole executive Tick value; there will often be a non-integer remainder. When this happens, the job cycle time in Ticks can be computed by either truncating the value or rounding the to the nearest whole integer Tick value:

$$T = n_\tau * \mathbf{trunc}(dt_\omega * M_e) \tag{5}$$

or

$$T = n_\tau * \mathbf{round}(dt_\omega * M_e) \tag{6}$$

For truncation using Equation 5 the cumulative Tick count is computed using the number of executive Ticks since the beginning of the simulation (n_τ) and the truncated value of the job cycle time (dt_ω) scaled by the executive multiplier (M_e). For rounding using Equation 6 the cumulative Tick count is computed using the number of executive Ticks since the beginning of the simulation (n_τ) and the rounded value of the job cycle time (dt_ω) scaled by the executive multiplier (M_e). The first three (3) seconds of an example simulation execution are shown below in Figures 4 and 5:

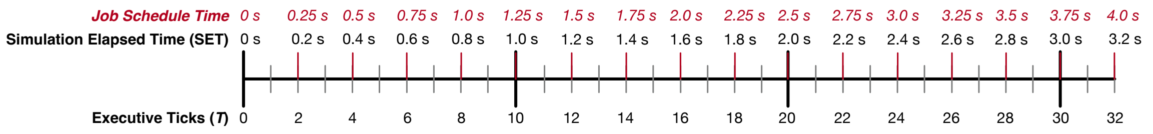

Figure 4: **Example Truncated Tick Job Scheduling.**

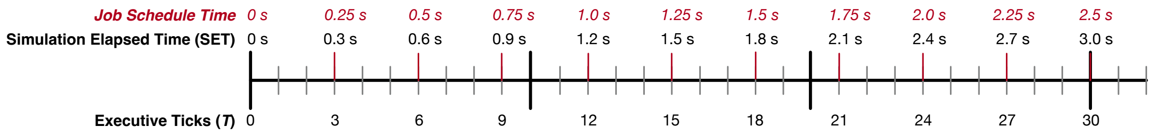

Figure 5: **Example Rounded Tick Job Scheduling.**

Note that in neither of these cases do the desired job schedule times ever align with the expected SET times. In the truncation case, the jobs are called more often than expected. In the rounding case, the jobs are called less often than expected. Again, in order for the job schedule times to align with the expected SET times, the job cycle times must be compatible with the executive time resolution.

2.6 Mapping to Scenario Time

SET usually works well for time-based systems that are time epoch independent. Typically, these are systems defined purely as functions of relative time steps (dt) and not absolute times (sometimes referred to as ephemeris times). SET is often insufficient for systems that require absolute physical time representations. For instance, planetary positions are usually defined with respect to well defined and precise astronomical time systems (e.g., Terrestrial Time). These cases require the definition of at least one Simulation Scenario Time (SST) time line.

Typically, an SST is computed as a direct linear relationship with respect to SET using a scaling of 1 and an offset as follows:

$$t_{ss} = t_{ss0} + t_{se} = t_{ss0} + (T/M_e) \tag{7}$$

where t_{ss0} is the simulation scenario time (SST) value corresponding to SET $= 0$, the beginning of the simulation execution. This is referred to as the SST epoch. The epoch value is dependent on the time standard(s) used to represent the simulation scenario(s) and the scenario start date/time.

Note that some commonly used time standards have more complicated relationships to a rate monotonic time advance. For instance, Coordinated Universal Time (UTC) has infrequent and unpredictable one second time adjustments called Leap Seconds to keep it coordinated with time standards based on the Earth's rotation. There are also relativistic time standard used for astronomical computations that are not linear with respect to Earth time. The reader in encouraged to refer to the SpaceFOM document for more information on time systems [5].

3 Real-Time Considerations

The executive cycle and time relationships expressed above provide an effective methodology for determining job scheduling frames and mapping those frames to the SET and SST time lines. However, they do not link the executive frames to either the Physical Time (PT) or the Computer Clock Time (CCT) time lines. In most cases, simulations do not need or even want to constrain the time advance of the executive to CCT. These simulations will execute as fast as the computational infrastructure will support and are sometimes referred to as non-real-time simulations. However, there are cases where constraining the executive to advance with real-time is necessary. These simulations are sometimes referred to as real-time simulations.

3.1 Time Management Options

In general, real-time execution refers to a simulation executive that is constrained by the progression of a computer clock or timing signal. However, there are a number of variations in how a simulation executive can manage the progression of CCT and the enforcement of time schedule boundaries. SpaceFOM categorizes real-time execution types into the following hierarchy [5]:

1. As-Fast-As-Possible (non-real-time)
2. Clock Constrained (real-time)
    a. Scaled Real-time
    b. Real-time
        i. Strict/Conservative Real-time (no frame overruns)
        ii. Elastic Real-time (catch-up on overruns)
            1). Limited Overruns
            2). Unlimited Overruns

This discussion is limited to general clock constrained real-time case (2.a). The other real-time options are variations on this case. All cases require mapping between execution frames in SET and a real-time frame in CCT.

3.2 Constraining to Real-Time

There are a number of ways to constrain a simulation executive run loop to real-time. The method discussed here is a simplified frame-based approach similar to that used in Trick [7]. The process starts with measuring the processing time dt_{fp} needed to execute all jobs for a given scheduled execution frame:

$$dt_{fp} = \sum_{i=0}^{n_j} dt_{ji} \tag{8}$$

where n_j is the number of jobs in a given frame and dt_{ji} is the processing time for each job in the frame. While this accounts for the processing time required by the scheduled jobs in the frame, there is also additional processing time used by the executive in various overhead tasks (dt_{eo}). The total processing time for the frame includes both and can be expressed by the simple equation:

$$dt_f = dt_{eo} + dt_{fp} \tag{9}$$

The next step is to associate these frame execution processing times with executive scheduling constraints derived from the CCT time line.

If dt_{rt} is the cycle time for the real-time frame, then the computation for the CCT used in a real-time frame is:

$$dt_{rt} = dt_{eo} + dt_{fp} + dt_{fm} \tag{10}$$

where dt_{fm} is the margin of CCT time left over in the real time frame after the jobs are processed and accounting for all executive overhead processing. This gives the available real-time frame processing margin as:

$$dt_{fm} = dt_{rt} - (dt_{eo} + dt_{fp}) = dt_{rt} - dt_f \tag{11}$$

Note that if $dt_{fm} \geq 0$ there is enough CCT time for all the job processing. If $dt_{fm} < 0$ the real-time frame is not sufficient to complete all required job processing.

While this quantifies the principal real-time frame requirements to maintain a real-time execution, it does not actually tie the execution to real-time (CCT). To tie the executive to CTT, assume there are no intervals where the executive goes into freeze, then the desired CCT at a real-time frame boundary can be computed as:[2]

$$t^*_{cc} = t_{cc0} + t_{se} = t_{cc0} + (T/M_e) \tag{12}$$

where t^*_{cc} is the desired CCT, t_{cc0} is the CCT at the start of the simulation, and t_{se} is the SET value for this real-time frame.

In order for a simulation to run in real-time, the real value of CCT must be constrained to equal the desired value of CCT:

$$t^*_{cc} = t_{cc} \quad \text{or} \quad dt_{cc} \equiv t_{cc} - t^*_{cc} = 0 \tag{13}$$

where dt_{cc} is the difference between the current CCT value and the desired CCT value.

This leads to two possible real-time states: $dt_{cc} \leq 0$ and $dt_{cc} > 0$. If $dt_{cc} > 0$ then the real-time deadline has already passed. That particular frame takes longer to execute than the real-time frame allows.[3] If $dt_{cc} \leq 0$ then the particular real-time frame can be met. The executive can pause the execution loop until $t_{cc} = t^*_{cc}$. Then resume the execution loop. This will constrain the execution loop to run in real-time or at least meet a real-time deadline on each real-time frame.

It is important to note that between real-time frames, the executive loop still runs unconstrained (as-fast-as-possible). This means that all jobs run unconstrained in the order determined by their cycle times and any ordering priorities determined by the executive. Any jobs in the intervening frames between real-time frames will run as fast as the processors can support. Typically, the executive goes into some kind of wait-loop after the last job in the real-time frame runs, checking for the appropriate CCT for that frame, and then releasing after reaching the scheduled CCT.

3.3 Constraints on the Real-Time Frame

How often should the executive check for real-time? In other words, what should be selected for a Real-Time frame (dt_{rt})?[4]

For well behaved real-time systems, the Real-Time frame should meet the same cycle time (frequency) frame scheduling constraints discussed in Section 2.4. This will ensure that the Real-Time frame is compatible with the executive Tick multiplier (M_e) selected for the simulation executive based on job execution frequencies.

However, there are additional considerations. For simulations that have jobs scheduled at multiple frequencies, the number and order that jobs execute in the real-time frame could vary depending on the selection of dt_{rt}. In fact, the only way that the number and order that jobs execute in the frame will be the same is if dt_{rt} is set to the Least Common Denominator of the integer factorization of the job cycle time ratios (M_p). In the examples in Section 2.4, this was 1920 for job frequencies of 128, 30, and 10 Hz. It was 9600 for job frequencies of 128, 100, 30, and 10 Hz. This would drive the dt_{rt} to be $1/1920s$ and $1/9600s$ respectively for those two cases. This is probably not practical.

Fortunately, the only non-negotiable requirement is that the dt_{rt} be compatible with the selected M_e. The value of dt_{rt} can be selected based on the real-time needs of the simulation and any time critical elements in the simulation. For instance, a simulation requiring human interaction may set the Real-Time frame to be 10 Hz ($dt_{rt} = 0.1\ s$). This ensures that the simulation maintains a real-time execution rate and supports human response times on the order of 0.1 seconds. However, a simulation connected to critical hardware running at 100 Hz should check at each cycle ($dt_{rt} = 0.01\ s$). This will help maintain the safety of the system.

3.4 Simple Real-Time Example

To help clarify the way jobs are interleaved between frames, we can explore a simple example. Assume 4 jobs with cycle times in seconds of 1.0 (1 Hz), 0.5 (2 Hz), 0.25 (4 Hz), and 0.125 (8 Hz) respectively. Using the prime

[2]Time periods where the simulation goes into freeze can be factored into the t_{cc} computations easily, but just add distractions for this discussion.

[3]This could be handled by one of the alternate real-time management options above.

[4]For Trick-based simulations, the Real-Time frame is the Software Frame.

integer factorization method discussed above will lead to a minimum M_p of 8 τ/s ($\epsilon = 0.125\ s$).

$$\begin{aligned}
1/8 &= 1/2^3 = 1/2^3 = 1/8 \\
1/4 &= 1/2^2 = 2/2^3 = 2/8 \\
1/2 &= 1/2 = 2^2/2^3 = 4/8 \\
1/1 &= 1/1 = 2^3/2^3 = 8/8
\end{aligned}$$

Assume that the chosen Real-Time frame is $dt_{rt} = 0.5\ s$ or 2 Hz. Given this job configuration, what jobs run on what frames and where are they with respect to the Real-Time frame?[5]

Note, that all job scheduling times are in SET. Assume that all simulation initialization jobs complete before entering into real-time. At SET = 0.0 all jobs will run because by definition everything synchronizes at zero.[6] Specifically, at SET = 0.0 s the 8 Hz, 4 Hz, 2 Hz, and 1 Hz jobs will run. At SET = 0.125 s only the 8 Hz jobs will run. At SET = 0.25 s the 8 Hz and the 4 Hz jobs will run. At SET = 0.375 s only the 8 Hz jobs run again.

At this point, all these jobs have been run sequentially by the executive without regard to CCT or real-time. The next schedule frame will be SET = 0.5 s. Note that the 0.5 s frame is the Real-Time frame. Here, the executive will synchronize SET with CCT by waiting on the actual CCT (t_{cc}) to catch up to the desired CCT ($t^*_{cc} = 0.5\ s$); specifically, that the real-time constraint has been satisfied ($dt_{cc} = 0$) (see Equation 13). Once SET time (SET = 0.5 s) is synchronized with the real-time frame (CCT = 0.5 s), the 8 Hz, 4 Hz, and 2 Hz jobs will run. At SET = 0.625 s only the 8 Hz jobs run again. At SET = 0.75 s the 8 Hz and 4 Hz jobs run again. At SET = 0.875 s only the 8 Hz jobs run again.

Here again, at SET = 1.0 s the executive will synchronize SET with CCT by waiting on the actual CCT (t_{cc}) to catch up to the desired CCT ($t^*_{cc} = 1.0\ s$). At this point, all jobs run again: 8 Hz, 4 Hz, 2 Hz, and 1 Hz. Note that 1 Hz is an integer multiple of the Real-Time frame; it will only run every other Real-Time frame. From this point on, the executive job scheduling and real-time checks repeat. See Figure 6 for a graphic representation of job scheduling, frames, and real-time effects.

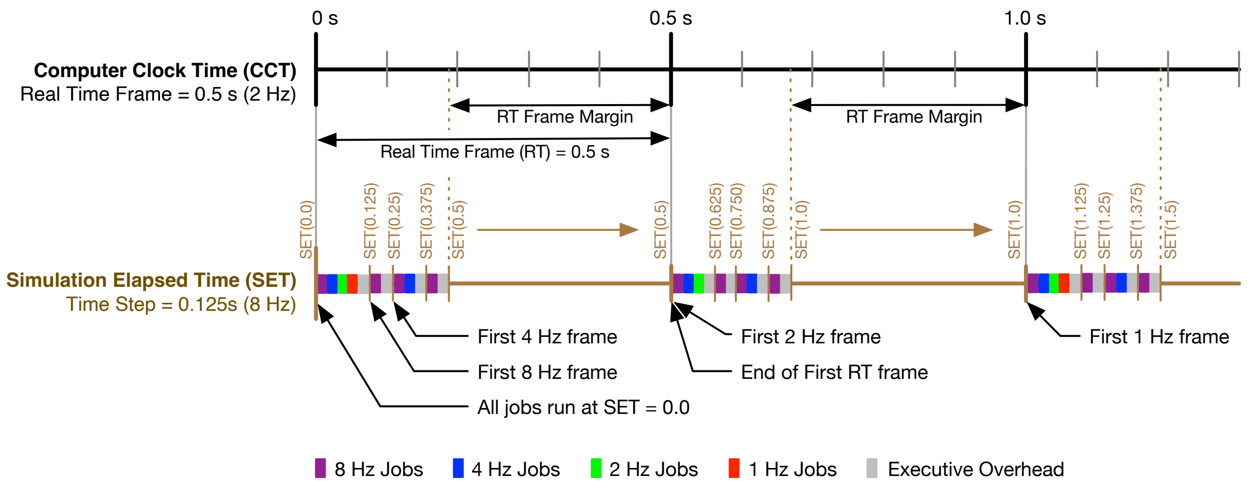

Figure 6: **Example Job Scheduling and Real-Time frame.**

Note that the minimum value of M_p that will work is 8 τ/s. This provides an executive time resolution of $\epsilon = 0.125\ s$. That is probably not a sufficient time resolution for the executive. Fortunately, any integer multiple of 8 will work and 1000 is an integer multiple of 8 that provides a time resolution of $\epsilon = 1\ ms$. A value

[5]Initialization job ordering and intra-frame job ordering are beyond the scope of this paper. Fortunately, they do not directly impact the general concepts of time compatibility and frame scheduling.

[6]For this discussion, we can ignore any offset job scheduling.

of $M_e = 1000$ translates to the following executive Tick intervals for the jobs:

$$\begin{aligned}
dt_8 &= 1/8\ s = (1/8)\ s * 1000\ \tau/s = 125\ \tau \\
dt_4 &= 1/4\ s = (1/4)\ s * 1000\ \tau/s = 250\ \tau \\
dt_2 &= 1/2\ s = (1/2)\ s * 1000\ \tau/s = 500\ \tau \\
dt_1 &= 1/1\ s = (1/1)\ s * 1000\ \tau/s = 1000\ \tau
\end{aligned}$$

4 Distributed Simulation

Over the years, a number of distributed simulation technologies have been developed. One of the more widely used and well-developed collaborative distributed simulation standards is the IEEE 1516 High Level Architecture (HLA). HLA provides a rich set of capabilities subject to documented limitations, restrictions, and constraints. These are enumerated in the *Framework and Rules* volume of the standard [2].[7] The HLA framework is formulated into a collection of constituent Management Services: Federation, Declaration, Object, Ownership, Data Distribution, and Time. Of those services, Time Management Services is probably one of the least understood and least used. Regardless, Time Management Services are critical to formulating HLA-based distributed simulations that maintain coherent data exchange and have repeatable execution results.

Real-time hardware-in-the-loop (HWITL) requirements further complicate the use of HLA Time Management Services. However, these are important requirements of many simulations and critical to real-time integrated verification and test simulations. HLA Time Management services provide the means for joined federates in an HLA-based federation execution to operate with a logical shared concept of time in the form of a distributed virtual clock. These services provide the infrastructure required to assure a causal ordering of time-based events. This is a requirement for time-based physics-based technical simulations like those used to support human space exploration activities.

4.1 HLA Logical Time

This leads to a discussion on how time is represented across a collection of coordinated and time synchronized HLA-based distributed simulations. The HLA standard uses a concept called HLA Logical Time (HLT). Each individual federate maintains its own local concept of HLT in addition to the SET, SST, and CCT time lines already mentioned. Through the HLA Time Management Services, these federate specific concepts of time progression can be linked together to form a federation execution wide progression of time.

Early versions of HLA did not have a 'standardized representation' for time [8]. However, modern versions of HLA support two principal standard types: 64-bit integer time and 64-bit floating-point time. These each have separate API interfaces. This paper will proceed using the **HLAInteger64Time** time representation. It is important to note that, like simulation executive Ticks, this time representation does not have any direct association with a 'continuous' time scale. The mapping between HLT and federate time scale is defined by the federation execution.

Not coincidentally, the integer HLT representation is similar to the executive Tick representation of time discussed in Section 2.4. As integer base representations, they both represent time in discrete integer quanta of time, essentially a Tick (τ). Where τ_f is defined as the executive Tick for a given federate f in the federation execution and τ_{hlt} is the HLT Tick for the collective federation execution.[8]

4.2 Time Ticks and Multipliers

The same relationship between simulation elapsed time t_{se}, simulation Tick count n_τ, and executive tick multiplier M_e expressed in Equations 1 and 2 applies to each individual federate in an HLA-based federation execution.

$$T_f \equiv n_\tau = t_{se} * M_f \quad \text{or} \quad t_{se} = T_f / M_f \tag{14}$$

[7]The HLA standard also includes two additional implementation documents: one that defines some standard language programming interfaces (APIs) [3]; and another that defines the data exchange standards and specifications [4].

[8]Note that τ_f is a the specific federate's τ from Section 2.4

where n_τ are the number of executive Tick cycles since the start of the federate and the definition of T_f. M_f is the executive Tick multiplier for a given federate. Specifically, each federate in a federation execution will have an executive multiplier M_f compatible with the jobs associated with that particular federate.

The reader my then question, what happens when all these federates execute in a time synchronized coordinated federation execution? As we saw in Section 2.4, adding additional jobs at different cycle times (frequencies) may require computing a different executive Tick multiplier. What does this mean for a distributed simulation composed of multiple federates (simulations)? Do we have to find a federation wide common prime factorization based on all the job cycle time in all the federates, say M_{hlt}?

The short answer is a qualified Yes! Fortunately, we may not necessarily have to account for the cycle time of all the jobs in all the federates in the federation execution. It is enough to account only for the job cycle times directly associate with HLA-based data exchange and services. The other jobs that are not directly associated with HLA-based data exchange and services are essentially independent from the HLA time synchronization requirements and only subject to the local constraints.

Let J_i be the set of all jobs in a specific federate:

$$J_i \in \text{Jobs in federate } i \tag{15}$$

Let J_{hla_i} be the set of all HLA related jobs in J_i:

$$J_{hla_i} \in \text{HLA related jobs in federate } i \tag{16}$$

Then J_{hla_i} is a subset of J_i:

$$J_{hla_i} \subset J_i \tag{17}$$

As a result, all the HLA related jobs in the federation execution J_{hla} are the union of all the sets of federate HLA related jobs J_{hla_i}:

$$J_{hla} \bigcup_{i=1}^{n_f} J_{hla_i} \tag{18}$$

where n_f is the number of federates in the federation execution. The federation wide Tick multiplier (M_{hlt}) is computed based on the prime factorization of the cycle times (frequencies) of all the jobs in set J_{hlt}.

At this point, an example might help. Assume that there are three federates in a federation execution: F_1, F_2, and F_3. Let F_1 have jobs with schedule frequencies of 1, 4, and 10 Hz. Let F_2 have jobs with schedule frequencies of 1, 2, 4, 10, and 100 Hz. Finally, let F_3 have jobs with schedule frequencies of 2, 10, and 30 Hz. This leads to the following independent federate executive Tick multipliers for federates F_1, F_2, and F_3 respectively: $M_1 = 10$, $M_2 = 100$, and $M_3 = 30$. However, the common prime factorization for all three federates combined would result in a federation Tick multiplier of $M_{all} = 300$.

Now assume that not all the jobs in F_1, F_2, and F_3 are associated with HLA activities. For instance, the 100 Hz jobs in F_2 may be for state integration and not directly related to an HLA data or service tasks. Let the HLA related job frequencies for the federates be: $F_{1hla}\{1, 4, 10\}$, $F_{2hla}\{1, 2, 4, 10\}$, and $F_{3hla}\{2, 10\}$. Note that for this combination the resulting common prime factorization results in $M_{hlt} = 20$ not the $M_{all} = 300$ for all the jobs.

It can be shown that the executive Tick multiplier for any given federate (M_f) in a federation execution must be greater than or equal to the associated federation execution HLA Logical Time (HLT) Tick multiplier (M_{hlt}):

$$M_f \geq M_{hlt} \tag{19}$$

While this is a necessary condition, it is not sufficient. The HLT Tick multiplier (M_{hla}) must also be compatible with every federate's executive Tick multiplier (M_f). Specifically, every M_f must be an integer multiple of M_{hlt} to ensure the simulation executive can support the HLA look ahead and data exchange times:

$$M_f \ \% \ M_{hlt} = 0 \tag{20}$$

where % is the Modulo operator.

Note that in the preceding example that neither of these conditions are meet for all federates. Specifically, $M_1(10) < M_{hlt}(20)$ and $M_3(30) \ \% \ M_{hlt}(20) \neq 0$. Does this mean that these federates cannot work together?

Not necessarily. The executive Tick multipliers derived in the example are the minimum Tick multiplier values (M_p). Fortunately, an integer multiple of the M_p can be used. If the federate executive Tick multipliers are chosen to be $M_1 = 60$, $M_2 = 100$, and $M_3 = 60$ and the federation HLT Tick multiplier is $M_{hlt} = 20$, then both conditions will be met.

This means that in order to include a simulation in an HLA-based federation execution, it may be necessary to modify the federate's executive Tick multiplier (M_f) to be compatible with the federation execution Tick multiplier (M_{hlt})!

Given that a federate's executive Tick multiplier and the federation execution's Tick multiplier will most likely differ, how can you convert to and from the federate's executive Tick-based time to the federation execution's time base? The following two equations provide these reciprocal relationships:

$$T_{hlt} = (T_f * M_{hlt})/M_f \tag{21}$$

$$T_f = (T_{hlt} * M_f)/M_{hlt} \tag{22}$$

where T_{hlt} is the integer HLA Logical Time (HLT) in the M_{hlt} base time units (τ_{hlt}) and T_f is the federate integer time in M_f base time units (τ_f). Remember that Equation 14 provides the means to compute the equivalent federate specific SET time, where $T \equiv T_f$ for the federate.

A final observation is that all time referenced federate simulation events (t_{event}) can only be represented as HLT times compatible with the federation execution's Tick multiplier (M_{hlt}). Note that from Equations 19 and 20, M_f/M_{hlt} will always be a whole integer and can be denoted as:

$$R_f \equiv M_f/M_{hlt} \tag{23}$$

Then Equation 22 can be rewritten without loss of generality as:

$$T_f = T_{hlt} * R_f \tag{24}$$

Now taking the modulo of both sides with respect to R_f results in:[9]

$$T_f \ \% \ R_f = ((T_{hlt} \ \% \ R_f) * (R_f \ \% \ R_f)) \ \% \ R_f \tag{25}$$

Note that $R_f \ \% \ R_f \equiv 0$ and that $0 \ \% \ R_f \equiv 0$. This reduces the preceding equation to just:

$$T_{f\_event} \ \% \ R_f = 0 \tag{26}$$

where T_{f_event} is the time of an event expressed in federate executive time Ticks (T_f). This implies that any compatible referenced federate simulation event must have a federate executive time Tick representation that is an integer multiple of the ratio of the federate to federation Tick multipliers (R_f).

For the example above, this results in multiplier ratios of $R_1 = 3$, $R_2 = 5$, and $R_3 = 3$. As long as an event time (T_{f_event}) is an integer multiple of R_f, the event will be compatible with the federation timing requirements.

5 Conclusions

The principal objective of this paper is to explore, understand, and quantify key factors for time representations in time-based simulations. Specifically, the underlying simulation executive time representation must be compatible with the cycle times (frequencies) that jobs will be called by the executive. Three principal area where explored: executive time representations in Section 2, real-time considerations in Section 3, and the time resolution relationships for distributed simulations in Section 4.

Here are the key observations drawn from these discussions:

- Executive time resolution (M_e) should be compatible with job cycle times (dt_ω).
- Prime factorization can be used to compute compatible base time resolutions: M_p.

[9] Note that $(a * b) \ \% \ n \equiv ((a \ \% \ n) * (b \ \% \ n)) \ \% \ n)$ [13].

- A compatible M_e will be any integer multiple of M_p.
- A simulation real-time frame should be compatible with the executive M_e.
- HLA-based distributed simulations also have a time resolution: M_{hlt}.
- HLA-based federate time resolutions must be compatible with the federation time resolution:
  - $M_f \geq M_{hlt}$
  - $M_f \ \% \ M_{hlt} = 0$
- HLA federate time events must be compatible with federation time resolution: $T_{f\_event} \ \% \ R_f = 0$.

Much of the content of this paper has been developed by engineers in the Software, Robotics, and Simulation Division (ER) at NASA's Johnson Space Center. ER develops and maintains a number of NASA open source simulation software packages including a simulation development environment called Trick [7] and an HLA/SpaceFOM middleware package called TrickHLA [14]. The time constraint checks described in this paper are being incorporated into both the Trick and TrickHLA initialization code to ensure time compatibility.

Acronyms

API	Application Programming Interface
CCT	Computer Clock Time
ER	The Software, Robotics, and Simulation Division
FOM	Federation Object Model
FST	Federation Scenario Time
HITL	Human-In-The-Loop
HLA	High Level Architecture
HLT	HLA Logical Time
HWITL	Hardware-In-The-Loop
Hz	Cycles per second
IEEE	Institute for Electrical and Electronics Engineers
JSC	Johnson Space Center
LCD	Least Common Denominator
M&S	Modeling and Simulation
NASA	National Aeronautics and Space Administration
PT	Physical Time
SET	Simulation Elapsed Time
SISO	Simulation Interoperability Standards Organization
SST	Simulation Scenario Time
SpaceFOM	Space Reference Federation Object Model

References

[1] E. Crues and D. Dexter. A Discussion of Time Management Concepts and Time Constraint Equations for Multi-Rate Federation Executions. In *SISO 2025 Simulation Innovation Workshop*. SISO, SISO, February 2025.

[2] Simulation Interoperability Standards Organization/ Standards Activities Committee (SISO/SAC). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*. Technical Report IEEE-1516-2010, The Institute of Electrical and Electronics Engineers, 2 Park Avenue, New York, NY 10016-5997, August 2010.

[3] Simulation Interoperability Standards Organization/ Standards Activities Committee (SISO/SAC). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification*. Technical Report IEEE-1516.1-2010, The Institute of Electrical and Electronics Engineers, 3 Park Avenue, New York, NY 10016-5997, August 2010.

[4] Simulation Interoperability Standards Organization/ Standards Activities Committee (SISO/SAC). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification*. Technical Report IEEE-1516.2-2010, The Institute of Electrical and Electronics Engineers, 3 Park Avenue, New York, NY 10016-5997, August 2010.

[5] E. Crues, D. Dexter, A. Falcone, A. Garro, M. Madden, B. Möller, and D. Ronnfeldt. *Standard for Space Reference Federation Object Model (SpaceFOM)*. Number SISO-STD-018-2020. Simulation Interoperability Standards Organization (SISO), P.O. Box 781238, Orlando, FL 32878-1238, USA, October 2019.

[6] Edwin Z. Crues, Dan Dexter, Alberto Falcone, Alfredo Garro, and Björn Möller. SpaceFOM - A robust standard for enabling a-priori interoperability of HLA-based space systems simulations. *Journal of Simulation*, 16(6):624–644, 2022.

[7] National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch (ER7). Trick simulation environment: Documentation. Trick Wiki Site: `https://nasa.github.io/trick/documentation/Documentation-Home`, September 2023.

[8] M. Karlsson, F. Antelius, and B. Möller. Time Representation and Interpretation in Simulation Interoperability - An Overview. In *SISO 2011 Spring Simulation Interoperability Workshop*. SISO, SISO, April 2011.

[9] Department of Defense Modeling and Simulation Coordination Office. *Modeling and Simulation (M&S) Glossary*. https://www.msco.mil, 1901 N. Beauregard St., Suite 500, Alexandria, VA 22311, October 2011.

[10] IEEE C/MSC - Microprocessor Standards Committee. *IEEE Standard for Floating-Point Arithmetic*. Technical Report IEEE-754-2019, The Institute of Electrical and Electronics Engineers, 2 Park Avenue, New York, NY 10016-5997, June 2019.

[11] Wikipedia. Integer factorization. Wikipedia: `https://en.wikipedia.org/wiki/Integer_factorization`, November 2025.

[12] Eric W. Weisstein. Prime factorization. MathWorld: `https://mathworld.wolfram.com/PrimeFactorization.html`, December 2024.

[13] Wikipedia. Modulo. Wikipedia: `https://en.wikipedia.org/wiki/Modulo`, November 2025.

[14] National Aeronautics and Space Administration, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch (ER7). TrickHLA. TrickHLA GitHub Site: `https://github.com/nasa/TrickHLA`, January 2025.

Author Biographies

Edwin Z. {Zack} Crues: *received B.S., M.S., and Ph.D. degrees in Aerospace Engineering from the University of Texas, Austin in 1983, 1985, and 1989 respectively. Zack has over 35 years of professional experience in developing spacecraft simulation and simulation technologies internationally. Zack currently works at NASA Johnson Space Center in Houston, Texas as a Space Systems Simulation Architect in the Simulation and Graphics Branch where he leads the development of simulation technologies and the application of those technologies in the simulation of NASA's current and proposed human space exploration missions.*

Daniel E. Dexter: *is an engineer in the Spacecraft Software Engineering Branch in the Software, Robotics and Simulation Division of the Engineering Directorate at NASA's Johnson Space Center in Houston, Texas. Daniel has over 30 years of software and simulation development experience including the areas of digital signal processing, image processing, neural networks, distributed supercomputing, embedded flight software, and distributed simulations. Daniel is the principal developer of the TrickHLA software package, a NASA developed IEEE 1516 simulation interoperability middleware for the NASA Trick simulation environment M&S tool.*