Characterizing, Measuring, and Validating the Temporal Consistency of Live–Virtual–Constructive Environments

Douglas D. Hodson
ASC Capabilities Integration Directorate
Wright-Patterson AFB, OH, USA

Rusty O. Baldwin
Air Force Institute of Technology
Wright-Patterson AFB, OH, USA
rusty.baldwin@afit.edu

A distinguishing characteristic of interactive live–virtual–constructive (LVC) environments is the relaxation of data consistency to improve the performance and scalability of the underlying distributed simulation. Relaxing data consistency improves the interactive performance of the environment because autonomous distributed simulation applications can continue executing and responding to local inputs without waiting for the most current shared data values. Scalability also improves since live and simulated entities from distant geographic locations can be interconnected through relatively high-latency networks. We introduce a temporal consistency model to formally define consistency for the dynamic shared state of a LVC environment for both continuous and discrete data objects. The level of inconsistency tolerated by a LVC is found to be a function of the accuracy and timeliness requirements for the distributed data objects. These requirements are mapped to specific time intervals for which data objects are considered valid. We also develop a real-time algorithm to compute the temporal consistency of individual data objects within the LVC.

Keywords: live, virtual, constructive, simulation, real-time, temporal consistency

1. Introduction

The Department of Defense has defined three distinct classes of military simulation: live simulation, virtual simulation, and constructive simulation. In a live simulation, real people operate real systems. For example, a pilot launching weapons from a real aircraft at real targets for the purpose of training, testing, or assessing operational capability is a live simulation. In a virtual simulation, real people operate simulated systems or simulated people operate real systems. For example, a pilot flying a simulated aircraft, launching simulated weapons at simulated targets would be considered a virtual simulation. In a constructive simulation, simulated people operate simulated systems.

SIMULATION, Vol. 85, Issue 10, October 2009 671–682

DOI: 10.1177/0037549709340732
Figures 1, 3–8, 12, 13 appear in color online:
http://sim.sagepub.com

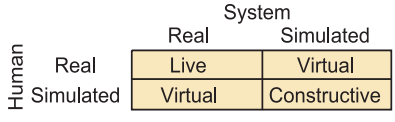

Figure 1. Simulation classification framework

Figure 1 provides a useful framework to classify these simulations based upon the types of entities they include. Entities in a live simulation include real people and real systems. Entities in a virtual simulation include simulated systems operated by real people. The entities in a constructive simulation are completely simulated by computer models and are often referred to as 'computer generated forces'.

While categorizing simulations into three distinct classes is useful, in practice it is problematic because there is no clear division between these categories: the degree of human participation in the simulation is infinitely variable, as is the degree of system realism [1]. Owing to this,

many simulations are actually hybrid systems that contain a mix of entity types. This is particularly true for virtual simulations which routinely include both virtual and constructive entities.

Live–virtual–constructive (LVC) simulations broaden the scope of virtual simulations even further by incorporating live assets into the interactive environment. To create a context for the environment, a hybrid simulation is assembled from a collection of autonomous distributed simulation applications which we refer to as a 'LVC'. Within the LVC, individual entities, vehicles and weapon systems are generated by specific simulation applications responsible for sharing current state information through a network infrastructure.

In a LVC, the 'system under study' is often a collection of 'system of systems' which includes humans and/or real operational system hardware. As these real-world elements are present, timing constraints are imposed on the simulation environment which, from a software standpoint, places the simulation into the class of real-time systems.

LVCs execute by passing state data between distributed simulation applications. As a result, a fundamental conflict arises in LVCs; simulation applications require state data that is not locally managed to produce correct outputs. The conflict arises because, in many situations, the application cannot wait for the most current data value and still meet real-time interactive response time constraints. If the distributed processes are connected via a network infrastructure with a relatively high latency, data transmitted by one application might be considered inconsistent or 'too old' by the time it is received. This inconsistency in state data is a distinguishing characteristic of LVCs which must be recognized and managed to harness the realism and power LVCs can provide.

As LVCs are used for training, test and evaluation, experimentation, and strategy evaluation, this inconsistency must be controlled to maintain the validity of the environment with respect to its purpose. This paper characterizes this inconsistency, and provides a framework to define consistency requirements relative to the objectives of the system.

This paper is organized as follows. In Section 2 we review notions of simulation time and state, including how it is updated and managed in a LVC simulation. In Section 3, the relationship between performance, consistency, and its sources is discussed. Section 4 introduces a temporal consistency model for LVC environments and classifies distributed state data as either continuous or discrete. This classification defines validity intervals based upon accuracy and timeliness requirements for each shared data object. In Section 5, a Petri net is used to model a representative LVC to reveal fundamental properties and characteristics. Section 6 presents a useful algorithm to compute the consistency of distributed state data while Section 7 introduces a general result for estimating it. Finally, in Section 8, we discuss the practical application of temporal data consistency concepts to evaluate a candidate system design.

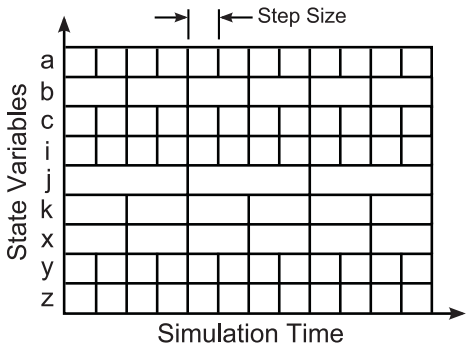

Figure 2. Time-stepped state space

2. Time and State

Fundamental to any notion of temporal consistency is time, and simulations have several notions of time. *Simulation time* is an abstraction used to model physical time. It represents elapsed time since the simulation started. For example, a floating-point variable could be used to measure time where 0.0 corresponds to the beginning of simulation execution. Time advances according to the particular simulation requirements, but not in real-time. *Wallclock time* refers to elapsed time. From the perspective of real-time system theory, wallclock time is *real-time*. It is continuous and linear. In a LVC (or real-time) simulation, the goal of the virtual and constructive components is to ensure *simulation time* advances and stays in synchronization with the wallclock. Clearly then, a LVC simulation is a real-time system.

A modeled system can be viewed as a collection of variables necessary to describe the system's state at a particular time, relative to the objectives of a study [2]. As simulation time advances, state variables change based on the time advance mechanism and can be represented as a state space diagram.

In a virtual simulation, human operators are often controlling vehicles where the position, velocity, and acceleration cannot be known in advance. Owing to this and because the vehicles themselves are often described by physics-based models, simulation time advances in fixed time steps closely synchronized with the wallclock.

In a fixed time-stepped simulation, state space variables are updated at regular intervals as shown in Figure 2. A state variable is not necessarily modified at each time step, but when a state variable is modified, it occurs at one of these intervals. As the figure shows, some variables are updated every other step, or even every fourth step depending upon the modeling requirements. The duration of the *step size*, as shown in Figure 2, is based on the na-

ture of the specific activity or activities system developers consider important [3].

2.1 Dynamic Shared State

Distributing a time-stepped simulation across multiple computers can be viewed as a partitioning of the state space among two or more autonomous simulation applications, where each application is responsible for maintaining its own local state and replicating state space data required by, but managed by other applications. This 'dynamic shared state' [4] constitutes the information that multiple simulation applications must maintain about the distributed environment.

Typically, simulation applications execute autonomously and asynchronously with respect to each other. That is, each application executes as an independent time-stepped simulation with its own state space. Some degree of interaction always takes place in the form of sharing state space information. If this were not the case, the LVC would no longer be considered distributed, it would simply be a stand-alone independent simulation. This sharing of locally managed state space information is achieved by sending messages through a network infrastructure.

Communication between applications is facilitated by a number of interoperability protocols and application programming interface (API) standards to so-called 'middleware'. Middleware is connectivity software with a set of enabling services that allow multiple simulation applications to interact across a network. The Distributed Interactive Simulation (DIS) [5] is a good example of a well-established protocol, whereas, the Data Distribution Service (DDS) [6], the High-Level Architecture (HLA) [7] and the Test and Training Enabling Architecture (TENA) [8] are representative middleware solutions.

3. Performance versus Consistency

A fundamental conflict arises in LVCs when the executing simulation applications require state data that is not managed locally, yet must also respond to inputs and produce correct outputs in real-time. The conflict arises because of network latency. If the network exhibits a relatively high latency, data transmitted by an application might be inconsistent or 'too old' when it is received.

Even so, to improve the performance and scalability of such systems, the state space of simulation applications is purposely allowed to become inconsistent. Performance is improved because each application continues executing and responding to local inputs without waiting for 'consistent' data to arrive. As data consistency is relaxed, scalability improves since, in general, this allows more applications from more distant geographic locations to be interconnected. Thus, portions of the state space within any particular simulation, at a given point in time, will not necessarily be consistent with the true state.

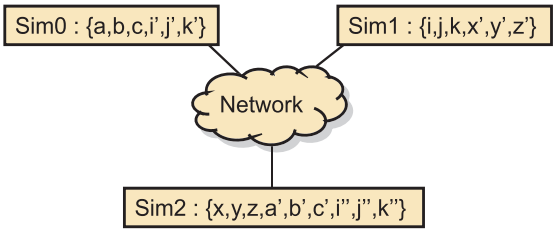

Figure 3. Distributed state space

Figure 3 shows a LVC composed of three simulation applications where Sim0 locally manages state space variables $\{a, b, c\}$ and replicates the state space variables managed by Sim1, namely $\{i, j, k\}$ as data is received from the network. Simulation Sim2 manages its own state space $\{x, y, z\}$ and replicates the state space of both of the other applications.

As the system is executing in real-time and synchronized to a wallclock, replicated state spaces often contain data that is older or 'aged' compared with the most current value. For example, since Sim0 is receiving updates via a network, its perception of Sim1's state space is inconsistent with Sim1's true state (i.e. it lags behind) because simulation time advances in lock step with the wallclock and cannot be stopped or paused to wait for an update to ensure consistency. As the local state variables of Sim1 change, it takes a finite amount of time to update Sim0.

Depending upon modeling and simulation requirements, this inconsistency might be acceptable. In fact, a DIS compliant simulation almost always works with aged data. As an example, the position of dynamic entities moving in the simulation rarely corresponds with the true position. The DIS standard specifies that for 'loosely' coupled interactions, entity position (state) updates received within 300 ms of when they are sent are acceptable. To reduce network traffic, updates are not sent on a regular basis; they are only sent when certain error thresholds are exceeded. The 'old' or aged data is considered good enough for the receiving application to estimate the true state values.

The level of inconsistency tolerated is based upon the accuracy of the estimation that can be made with older data and, second, the underlying requirements of the simulation itself.

3.1 Sources of Inconsistency

For a typical LVC simulation, sources of data inconsistency are introduced by the architecture of the distributed simulation applications and the interconnecting network infrastructure. We call the combination of the simulation architecture and the communication architecture the *system design*. We model the architectural characteristics of both the simulation applications and the communication

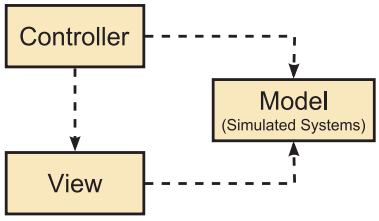

Figure 4. Multi-threaded MVC pattern

mechanism to quantify and estimate their effect on data aging.

3.1.1 Simulation Applications

The model-view-controller (MVC) architectural pattern is a well-established structure representative of the design of a typical simulation application. A central feature of the design pattern is the separation of user interface logic from the 'domain logic' which performs calculations and stores data.

In the context of virtual and constructive simulation applications, the domain logic are the simulated systems and state variables, while the graphical displays and input/output (I/O) functionality represent the view and controller components as shown in Figure 4. For live simulations, the domain logic often represents the periodic sampling of state information from live assets for the distributed environment.

The controller in the MVC pattern is typically associated with processing and responding to events that induce state changes in the model. Using this approach, the construction of a simulation application would, ideally, consist of a loop that, for each frame, sequentially reads inputs, executes system models, and generates outputs (i.e. updates graphics and processes network activities). In other words, this pattern is typically implemented as a single-threaded application whereby only the model (i.e. domain logic) makes state changes in response to controller events.

Doing this in real-time, however, is limited by processing power. This limit becomes ever more problematic as frame rates increase, thus reducing the amount of time available to complete all tasks (i.e. model updates, graphic drawing, and the processing of network activity). To further complicate matters, in the domain of a LVC, the latency associated with state data moving through the network imposes an upper bound on frame rates if data is to have some level of consistency across applications.

To resolve this fundamental conflict, a multi-threaded variant of the MVC pattern is often used [9] where the processing of models, graphics, and network I/O is done in separate threads, either synchronized with each other, or purposely allowed to execute asynchronously at assigned priorities. Typically, system state is executed by a high-priority thread, while graphics and network threads are set to a lower priority. This multi-threaded variant MVC pattern improves system response time with respect to a human participant, but at the cost of data consistency.

3.1.2 Interoperability Communication

LVC simulations are often implemented using dedicated networks to control utilization and provide enhanced data security. Controlling utilization is an important aspect of characterizing LVCs because it is assumed that network utilization has little or no effect on network latency. This is true as long as the network is not operating at high utilization rates exceeding 50–60%.

Since LVCs are time-sensitive applications, they often transmit state data using connectionless, efficient, transmission protocols, such as the User Datagram Protocol (UDP). As state space data updates are broadcast regularly, reliability is sacrificed to reduce overall latency. Furthermore, UDP and other connectionless protocols support the simultaneous distribution of state data using unicast, multicast or broadcast addressing schemes.

As the interaction of LVCs is affected by network latency, and communication latency is increased by software complexity, middleware is often not the best solution to share state data in these environments [10]. Therefore, we initially restrict our attention to LVCs that share state data through well-defined protocols such as DIS. This restriction is relaxed later.

The DIS standard specifies a number of factors and performance requirements to minimize both network delay and network delay variance [5]. For example, simulation state data is encoded into formatted messages, known as protocol data units (PDUs) and exchanged using existing transport layer protocols; normally broadcast or multicast UDP. The size of DIS PDUs range from 80 to 200 bytes which is significantly smaller than the maximum transmission unit (MTU) of 1,500 bytes for Ethernet. This prevents packet fragmentation at the link layer.

Given these network environment characteristics, we model a dedicated LVC network as transporting state data according to well-defined latency distributions with adjustable mean and shape parameters.

4. Temporal Consistency Model

As LVC simulations often execute with inconsistent data, it is useful to characterize consistency in terms of correctness, as any notion of data quality or correctness depends on the actual use of the data. In other words, data sufficient or accurate enough for one application might be insufficient for another.

We first define absolute consistency, then apply temporal consistency concepts developed in [11], [12], [13], and [14] to the domain of LVCs as they provide a useful framework for defining system requirements in terms of correctness.

Definition 1. (Absolute consistency) *Given a shared data object, θ, the state is absolutely consistent at any time t, if and only if for all i, j, $1 \leq i, j \leq n$, $\theta_i(t) = \theta_j(t)$, where n is the number of nodes (i.e. simulation applications) in the LVC.*

We say that the LVC is absolutely consistent if and only if every θ is consistent. In other words, a LVC is absolutely consistent if and only if the value of the replicated data objects as managed by each simulation application within the distributed system is consistent at all times.

Temporal consistency relaxes absolute consistency by defining the correctness of a shared data object, θ, as replicated by autonomous simulation applications as a function of a time interval. That is, the value of a shared data object is accurate or valid for a period of time after being updated.

Definition 2. (Temporal consistency) *A shared data object, θ, is temporally consistent if its creation timestamp, θ_{TS}, plus the* validity interval*, θ_{VI}, of the data object is greater than or equal to current time t, i.e. $\theta_{TS} + \theta_{VI} \geq t$.*

This notion of consistency is generalized for a distributed simulation consisting of n nodes, where each node defines a specific validity interval, $\theta_{i,VI}$, so that for all i, $1 \leq i \leq n$, $\theta_{TS} + \theta_{i,VI} \geq t$. As LVCs are connected via non-deterministic networks, validity intervals include an acceptable reliability statistic (e.g. 95% of the time).

4.1 Derived Data Objects

Another notion of temporal consistency is 'relative consistency'. It defines the accuracy or validity of *derived data* in terms of the relative creation times of the set of data used to produce it.

Definition 3. (Relative consistency) *The set of data objects used to derive a new data object, θ, form a* relative consistency set*, R. Each set R has a positive validity interval, denoted by R_{VI}. A derived data object, θ, is relatively consistent if for all $\theta \in R$, $|\theta_x - \theta_k| \leq R_{VI}$ where k is the cardinality of R.*

Thus, temporal consistency is viewed as a *freshness* constraint and relative consistency is a *correlation* constraint [15].

By defining LVC correctness requirements in terms of validity intervals for the shared data objects, we address the inconsistency in shared state data directly. Since the inconsistency in shared state data is the distinguishing characteristic of LVCs affecting performance and scalability, the validity intervals of state data directly relate to system performance. Furthermore, relaxing data consistency by increasing validity intervals improve both performance and scalability.

4.2 Classifying State Data

To define validity intervals, it is useful to classify state data as either 'continuous' or 'discrete'. This classification is based upon what the data represents or *what* is being modeled, not the mechanics of how its updated or processed by a digital computer. For example, the state describing the position of an aircraft in Cartesian coordinates would be considered continuous data, even though it is updated by a producer at some fixed frequency. The state data describing the position of a light switch would be considered discrete. After classifying state data, the determination of an appropriate validity interval defining LVC correctness is made.

For continuous objects, validity intervals establish correctness by bounding the difference (or accuracy) between a producer's value and a consumer's value [16]. For discrete objects, validity intervals establish correctness based on timeliness; the interval specifies that a consumer may never be out of synchronization with the producer by more than a validity interval time [16]. In other words, if the modeled system changes state, a consumer will receive the change no later than the time specified by the validity interval; until that time, the replicated state within the consumer is simply incorrect. The impact of temporally incorrect discrete state data is especially important and should be carefully considered as the specified validity establishes 'how long' incorrectness can be tolerated.

5. Simulation Properties

To understand the properties of a LVC, a conceptual model is built by abstracting the essential architectural features of simulation applications and networks that affect state space data timeliness. These effects are captured using a colored Petri net, which is particularly well suited for modeling systems in which communication, synchronization, and resource sharing are important. Jensen provides a comprehensive discussion of the theoretical foundations, analysis methods and the practical uses of colored Petri nets [17].

For the LVC architectural model, 'colored' tokens include attributes to represent simulation state data and the timeliness of that data; 'places' contain tokens and 'transitions' model the temporal properties of a particular system design. Time associated with transitions is specified by fixed or stochastic distributions. A 'snapshot' of all places in the model constitutes the state of the system at a particular point in time.

When a transition creates a token, the token's 'creation time' attribute is set. When a token is *moved* to a place, a second time-stamp attribute called the 'arrival time' is set. Time-stamping tokens with both a creation and arrival

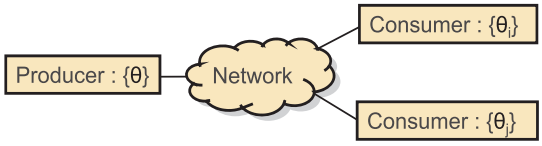

Figure 5. LVC model

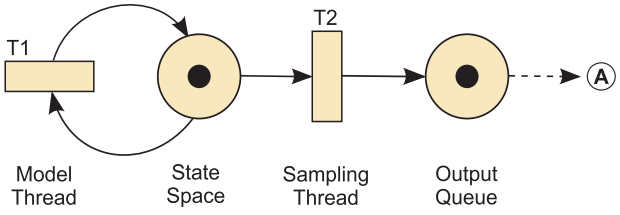

Figure 6. Producer model

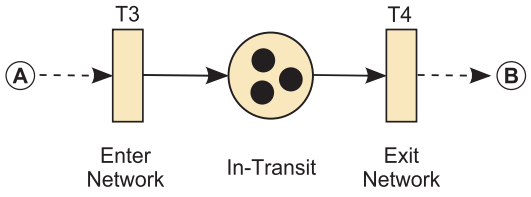

Figure 7. Network model

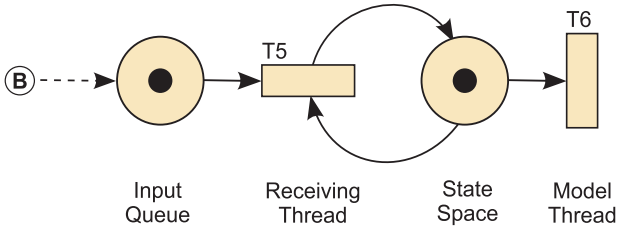

Figure 8. Consumer model

time captures the temporal dynamics of a particular system design. The temporal properties of transitions constitute fundamental limits in a system design and their effect on data timeliness are of great interest.

Figure 5 is an abstract model of a LVC with a producer application distributing state data θ to other applications or consumers of that data. For both producer and consumer (i.e. simulation applications within the LVC), we assume that the architectural organization and execution of the simulation architecture is mapped to the multithreaded MVC pattern discussed in Section 3.1.

Figure 6 is a Petri net model of the producer, where the simulated systems component from the MVC pattern periodically updates θ, while a thread servicing the network periodically transmits updates to consumers. Threads periodically updating and transmitting the various θ to consumers are asynchronous with respect to each other.

The model thread is represented by transition T1 that updates θ by replacing old state data (represented by a token) with new state data. The creation time attribute for this new token is set to the current simulation time. The state space place holds state tokens. It represents local computer memory that stores θ. The sampling thread is represented by transition T2 which models the asynchronous reading of θ for transmission to other applications. This transition copies tokens from computer memory to the output queue. The output queue holds data to be transmitted by the network infrastructure. This queue could also represent a graphical display in the multithreaded MVC pattern. The frequencies assigned to transitions T1 and T2 are the factors of most interest in a system design.

Figure 7 models the network which moves θ from producer to consumer with a particular latency. Transition T3 captures the temporal characteristics of a network infrastructure in terms of a distribution, such as exponential or Pareto with location and shape parameters. The location represents the mean latency of moving the data from a producer to a consumer. Each data packet is assumed to be the same size and the latency includes sender overhead to submit and transmit the message, the time of flight (propagation time), the transmission time (message size divided by bandwidth), and receiver overhead.

The enter network transition time records the time that tokens entering the network will arrive at their destination. This time is stochastically drawn from the abovementioned distribution and added to current simulation time. The exit network transition fires as soon as its enabling condition is satisfied; the moment simulation time advances to the earliest arrival time of any token at in-transit. The distribution assigned to T3 is also a factor of interest in the system design.

Figure 8 models a consumer receiving θ. This model captures the essential feature of the receiving thread: the update mechanism for the dynamic shared state.

The input queue stores tokens arriving from the network. The receiving thread is represented by transition T5 which is enabled when the input queue has tokens. In this model, 'old' state data already located in state space place is replaced with 'newer' data from the input queue the instant it arrives. In practice, transition T5 is implemented as a blocked thread waiting for data to arrive. The state space place is a combination of data received from other producers and data locally managed by the consumer. The consumer's model thread is represented by transition T6 which executes periodically.

Since T5 is enabled and fires immediately when tokens arrive for processing, it is not a factor of interest. Transition T6 on the other hand, determines when state data is

used for consumer calculations, and is therefore a factor of interest.

5.1 Startup Dynamics

The temporal characteristics of transitions T1, T2, T3, and T6 affect the consumers replicated state space data timeliness and consistency. Since producers, consumers, and their respective threads do not synchronize, but are executed on a periodic basis, there is a phase relationship between each periodically executed transition. To capture this property of a LVC, the phases ϕ_1, ϕ_2, and ϕ_6 associated with transitions T1, T2, and T6, respectively, are modeled as random variables in the system. As the phase relationship is relative, the model is simplified by arbitrarily selecting one phase, and setting it to zero. After some analysis we set $\phi_2 = 0$ as this leads to clearer insight and intuition into the roles and relationship each factor contributes to data aging.

Preliminary exploration of the LVC system model with randomly assigned phases yielded important insight that led to a simpler model for simulation and analysis. For example, the sampling of the producer's state space data by T2 for distribution to consumers is equivalent to the periodic creation of a new token in the output queue the age of which is drawn from a uniform distribution that ranges from zero to the period of the producer's model thread T1. Also, the randomly assigned phase associated with transition T6 can be simplified by using a uniform distribution to determine *when* the consumer's model thread uses state space data. The time associated with using state space data ranges from zero to the period of the consumer's model thread T6.

5.2 Analysis and Results

A simulation of the Petri-net-based LVC system model was used to vary the factors that affect the temporal properties. This includes the frequency of the producer's model and sampling threads, network latency characteristics as defined by a representative distribution, and the frequency of the consumer's model thread.

Exploration of the system model leads to the following two hypotheses concerning system dynamics. The first is that the period of the consumer's model thread, T6, does not influence data aging. In other words, the frequency at which the consumer uses replicated state data has no effect on the mean age and variance of that data. The second is that the mean age of the data used by a consumer in a LVC system can be estimated by adding each factors individual contribution to aging. In other words, given that a LVC is a linear system, each factor's contribution to aging is independent, and there are no interaction effects.

To validate these hypotheses, a preliminary two-level full factorial screening experiment, as shown in Table 1, was performed to determine each of the four identified factor's influence on data aging including any second-, third-, and/or fourth-order interaction effects. A two-parameter exponential distribution modeled network latency characteristics. For T3, Table 1 lists the location parameter for the distribution with a fixed standard deviation of ± 1 ms.

Table 1. Screening experiment

Factor	Levels
Producer model thread (T1)	50, 100 Hz
Producer sampling thread (T2)	5, 20 Hz
Network latency (T3)	5, 100 ms
Consumer model thread (T6)	50, 100 Hz

Table 2. Screening results

Number	T1 (Hz)	T2 (Hz)	T3 (ms)	T6 (Hz)	μ (ms)	σ (ms)
1	100	20	5	100	35	15
2	100	20	5	50	35	15
3	100	20	100	100	130	15
4	100	20	100	50	130	15
5	100	5	5	100	110	58
6	100	5	5	50	110	58
7	100	5	100	100	205	58
8	100	5	100	50	205	58
9	50	20	5	100	40	16
10	50	20	5	50	40	16
11	50	20	100	100	135	16
12	50	20	100	50	135	16
13	50	5	5	100	115	58
14	50	5	5	50	115	58
15	50	5	100	100	210	58
16	50	5	100	50	210	58

For each simulation run, the mean age and standard deviation of the replicated state data, as used by the consumer's model thread, was computed. The simulation terminating condition was reached when the mean age was within ± 1 ms of its true age with 95% confidence. Table 2 contains the results of the screening design for three replications of the experiment.

Comparing the mean and standard deviation for each pair of runs (e.g., 1 and 2, 3 and 4, etc.) in the table indicates that the consumer's model thread (T6) might not play a role in state space aging. This is confirmed by the analysis of variance (ANOVA) for mean data aging. Factors T1, T2, and T3 accounted for nearly all of the source of variation in aging with each having statistically significant p-values of zero. Factor T6 and just as importantly, all second-, third-, and fourth-order interaction terms played no role in explaining variance.

Table 3. Experimental design

Factor	Levels
Producer model thread (T1)	50, 80, 100 Hz
Producer sampling thread (T2)	5, 10, 20 Hz
Network latency (T3)	5, 50, 100 ms

This validates our first hypothesis and highlights an important characteristic of a LVC. From the standpoint of the consumer, the aging characteristics of the replicated state data is not influenced by the rate at which data is used. In other words, the frequency of the consumer's model thread (T6) does not play a role in the temporal characteristics of a LVC.

This somewhat counterintuitive result can be explained. Since there is no synchronization between the consumer's model thread and the rest of the system, a relative phase, ϕ, results. This mimics the actual startup conditions of a LVC. The random assignment is then, in effect, a sampling of the state space. Capturing this behavior was intentional, as it mimics a real-world distributed simulation where multiple asynchronous simulation applications are using the same state data, each at potentially different points in time.

After eliminating the consumer's model thread (T6) as a factor from the experiment, a second three-factor, three-level experiment, as shown in Table 3, was conducted to provide additional detailed data on each of the remaining factors contribution to aging including any second- and/or third-order interaction effects.

ANOVA results indicated that factors T1, T2, and T3 explain all of the variance in the LVC system model, and are statistically significant, each having a p-value of zero to three significant digits. Furthermore, each factor's contribution to state space aging is independent; there are no significant second- or third-order interaction effects. This validates our second LVC hypothesis; in terms of data aging characteristics, a LVC can be viewed as a first-order linear system.

5.3 Analytic Model

Using the results in Section 5.2, an analytic model to estimate the mean worst-case aging and variance of a system design can now be developed. In other words, an analytic model can be developed that considers all of the identified factors affecting data aging, including the initial startup dynamics described in Section 5.1, as well as the network latency characteristics. As LVCs are linear systems with respect to data aging, estimates can be made by adding each factor's contribution to age as follows.

A uniform distribution models the contribution to state space aging by model thread transition T1. The mean of this distribution is

$$\mu_{\text{Model}} = \frac{T_{\text{Model}}}{2} \tag{1}$$

where T_{Model} is the period of the model thread. State space age variance is

$$\sigma^2_{\text{Model}} = \frac{T^2_{\text{Model}}}{12}. \tag{2}$$

In a similar manner, the uniform distribution is used to characterize the contribution to state space aging by the sampling thread, T2. The mean of this distribution is

$$\mu_{\text{Sampling}} = \frac{T_{\text{Sampling}}}{2} \tag{3}$$

where T_{Sampling} is the period of the sampling thread. The contribution to age variance is

$$\sigma^2_{\text{Sampling}} = \frac{T^2_{\text{Sampling}}}{12}. \tag{4}$$

Using (1) and (3), the contribution of the producer's architecture to mean aging is

$$\mu_{\text{Producer}} = \mu_{\text{Model}} + \mu_{\text{Sampling}}. \tag{5}$$

Using (2) and (4), the contribution of the producer's architecture to variance is

$$\sigma^2_{\text{Producer}} = \sigma^2_{\text{Model}} + \sigma^2_{\text{Sampling}}. \tag{6}$$

The mean age of the replicated state space as seen by the consumer is

$$\mu_{\text{SS}} = \mu_{\text{Producer}} + \mu_{\text{Network}} \tag{7}$$

where μ_{Network} is the mean age due to network latency.

The variance associated with replicated state data aging is

$$\sigma^2_{\text{SS}} = \sigma^2_{\text{Producer}} + \sigma^2_{\text{Network}}. \tag{8}$$

Mean network latency, μ_{Network}, and variance, $\sigma^2_{\text{Network}}$, can be estimated using empirical data collected by tools such as ping, or the characteristics of a representative distribution. For example, a network modeled as by a two-parameter exponential distribution, would have a mean and variance contribution to aging of

$$\mu_{\text{Network}} = \gamma + \frac{1}{\lambda} \tag{9}$$

and

$$\sigma^2_{\text{Network}} = \frac{1}{\lambda^2} \tag{10}$$

respectively, where γ and λ are the location and scale parameters of the distribution.

Figures 9 and 10 compare the analytic model and simulation results for the mean worst-case and standard deviation of aging for several experimental design cases. For each design, the analytical model's predicted mean is within ± 1 ms with a 95% confidence interval of the simulation result.

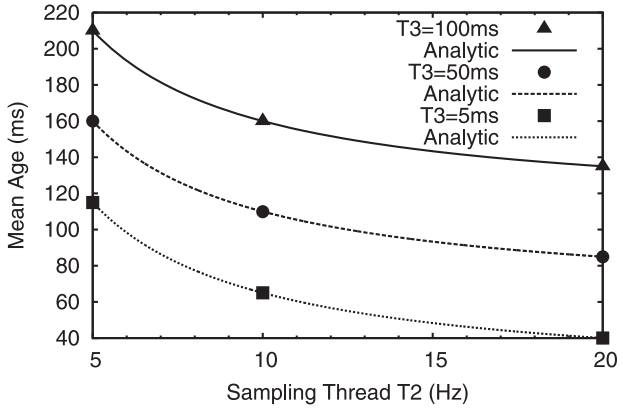

Figure 9. Mean worst-case age (ms) (T1 = 50 Hz)

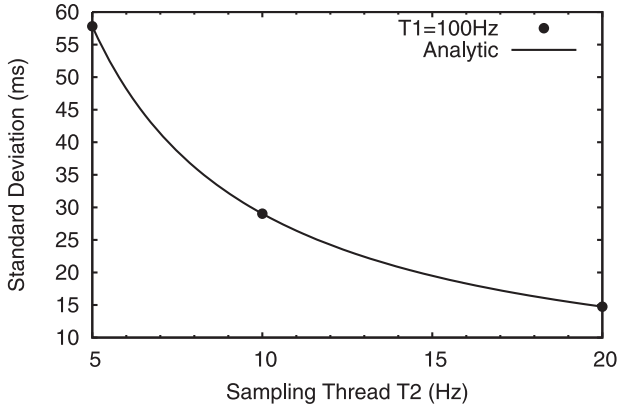

Figure 10. Standard deviation (ms) (T1 = 100 Hz, T3 = 5 ms)

6. Measuring Consistency

These simulation results lead to the development of an effective and efficient algorithm to calculate the mean age and variance as seen by the consumer. Data generated by the producer and placed into the consumer's state space can be viewed as shown in Figure 11.

That is, as a consumer receives data, its value is updated, but not necessarily with the most current value calculated by a producer's model thread (T1). The value received will, most likely, have aged due to the asynchronous sampling of the producer's sampling thread (T2) and network latency. Owing to this and the characteristics of stochastic non-deterministic networks, arriving data might be younger, or in some cases older, than the current value. Whatever the case, after its arrival, the state space ages linearly with wallclock time until the next update. We are interested in deriving the mean age and variance of state space data from the perspective of a consumers model thread.

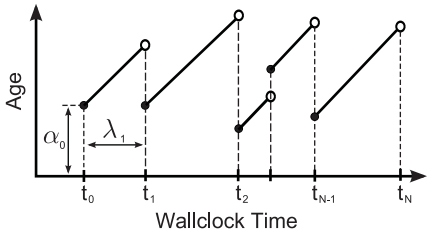

Figure 11. Distributed state space data

The mean of a function is the average value of the function over its domain. The mean μ of $f(t)$ over the interval (t_0, t_1) is

$$\mu = \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} f(t)\, dt. \tag{11}$$

We consider $f(t)$ the aging function associated with state space data and α to be its age upon arrival. As shown in Figure 11, $f(t)$ increases linearly from α_i to $\alpha_i + \lambda_{i+1}$, where λ is the update interval. In other words, data ages at the same rate as the wallclock time advances. Therefore, the aging function is

$$f(t) = t + \alpha_i \tag{12}$$

and the mean age can be calculated as a summation over all discrete intervals (i.e. the intervals defined by the interrarrival time between received updates) divided by total elapsed time. Thus, the mean age of the state space is

$$\mu_{SS} = \frac{1}{t_N} \left[\int_0^{t_1} f(t)\, dt + \int_{t_1}^{t_2} f(t)\, dt + \cdots + \int_{t_{N-1}}^{t_N} f(t)\, dt \right] \tag{13}$$

which after integration and simplification is

$$\mu_{SS} = \frac{1}{t_N} \sum_{i=1}^{N} \left(\frac{\lambda_i^2}{2} + \alpha_{i-1}\lambda_i \right) \tag{14}$$

where λ_i is the interrarrival time, and t_N is total elapsed wallclock time over N intervals.

The variance of state space aging is the mean square error minus the square of the mean. The mean squared error is

$$\text{MSE} = \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} f(t)^2\, dt \tag{15}$$

Using (12), (15) can be integrated and reduced to

$$\text{MSE} = \frac{1}{t_N} \sum_{i=1}^{N} \left(\frac{\lambda_i^3}{3} + \alpha_{i-1}\lambda_i^2 + \alpha_{i-1}^2\lambda_i \right) \tag{16}$$

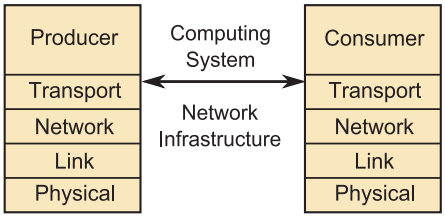

Figure 12. Latency classification and the OSI model

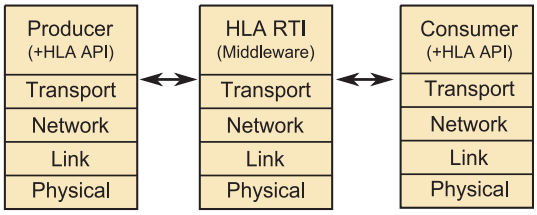

Figure 13. HLA-based communication.

Finally, the variance of the state space age is

$$\sigma^2_{SS} = \text{MSE} - \mu^2_{SS} \tag{17}$$

The continuous integration of data age as it is received is a practical way to compute data aging characteristics in the Petri net simulation. It is also a useful algorithm that can compute, in real-time, the temporal consistency of state data for a LVC.

7. Generalized System Model

As a LVC is a linear system, sources or generators of latency can be classified into general categories such as 'computing system', 'network', and 'middleware'. For the producer model in Section 5, factors T1 and T2 define the characteristics of a simulation application's computing system latency component while factor T3 defines the latency properties of the network modeled by an exponential distribution. This general concept is extended to include other potential sources of latency including middleware software such as HLA, DDS and TENA.

Figure 12 shows how computing system and network latency is classified with respect to the Open Systems Interconnection (OSI) Reference Model. For this model, the network infrastructure latency includes time consumed by all media, bridges, routers, gateways, encryption/decryption devices, and intervening networks. This is representative of a DIS-based simulation.

Figure 13 shows the software architecture of a HLA-based distributed simulation with respect to the OSI model. As HLA is an interface specification, the various implementations do not necessarily include a separate HLA run-time infrastructure (RTI) application. In this case, HLA API interfaces within the producers and consumers establish and manage all of the communication between the nodes. In others, the HLA API communicates with a separate RTI which manages all shared state data within the distributed simulation.

An estimate of data aging for a LVC system design that includes all of these sources is

$$\mu_{SS} = \mu_{\text{Computing}} + \mu_{\text{Middleware}} + \mu_{\text{Network}} \tag{18}$$

and the age variance of a system design is

$$\sigma^2_{SS} = \sigma^2_{\text{Computing}} + \sigma^2_{\text{Middleware}} + \sigma^2_{\text{Network}}. \tag{19}$$

8. Application

Given that managing LVC data consistency adds complexity to a system design, the motivation to interconnect geographically dispersed simulation applications can reasonably be questioned. To this point, we have considered a LVC system to be a set or collection of autonomous simulation applications, each designed to fulfill a specific role in the system design. While this might be ideal, more commonly, LVC systems are assembled by interconnecting a collection of *existing* simulation applications or by sampling real operational system hardware (live entities) which are often located at different geographic locations.

From a LVC development perspective, this is appealing, as much of the effort to create functional applications is complete. Unfortunately, this strategy limits the system architects ability to partition and tailor the set of dynamic shared states for maximum performance and scalability. An ideal partitioning scheme would involve sharing states with relatively large validity intervals.

Any proposed LVC system design should be considered as a 'candidate' to evaluate against defined consistency requirements based upon intended use and purpose. If the candidate system does not meet requirements, a new candidate should be derived that satisfies the consistency requirements.

8.1 Aerial Combat Example

As a practical example, consider a LVC system used to train fighter pilots for close-range aerial combat (i.e. dogfighting). The objective of the training is to learn tactical maneuvers which provide an advantage over an adversary. The LVC design connects two high-fidelity motion-based flight simulators located at different geographic locations.

Considering the positional accuracy requirements to conduct this training and the quality of dead-reckoning algorithms to estimate aircraft position, the system is said to be temporally consistent (i.e. correct) if shared data is no older than 140 ms, 98% of the time. As a point of reference, the DIS standard specifies a transport-to-transport (i.e. network infrastructure) latency value of 100 ms with an acceptable reliability of 98% for 'tightly' coupled interactions [18].

Table 4. Computing system worst-case analysis

Factor	Upper limit	Maximum (ms)
Model Thread	Bounded	20
Sampling Thread	Bounded	100
Total	Bounded	120

8.1.1 Candidate System Design

The specifications for the candidate LVC system are as follows.

- Two high-fidelity motion-based flight simulators connected across a dedicated network with a mean latency of 50 ms and a standard deviation of ± 5 ms.
- The dynamic shared state consists of continuous data objects that specify the position of each fighter.
- The local state space managed by each simulation application is updated at 50 Hz conforms to the producer system model presented in Section 5.
- Simulation applications transmit shared state updates at 10 Hz.

8.1.2 Evaluation

This candidate system design is evaluated by considering each source or latency generator's contribution to aging as presented in Section 7.

As shown in Table 4, computing system latency is determined by considering the startup dynamics (relative phasing relationship) between the modeling and sampling threads for the producer as described in Section 5.1. For this scenario, worst-case aging occurs when the relative phase between the threads is such that the sampling and periodic transmission of updates uses data from the producers state space already aged 20 ms. As the next periodic transmission does not occur for another 100 ms, it is possible the consumer will receive data at least 120 ms old. After including the network latency component of 50 ms, ± 5 ms, an estimated mean for data aging, μ_{SS}, is 170 ms. Thus, this candidate design does not meet requirements.

This is resolved by increasing the periodic rate at which state space updates are transmitted to 20 Hz which reduces the worst-case computing system latency to 70 ms and results an in overall estimate for μ_{SS} of 120–139 ms with 98% reliability. At this new rate, the LVC is now within bounds of the consistency requirements.

9. Conclusion

Since the first simulation networks of the early 1980s, to the current state-of-the-art in high-performance distributed computing and gaming, a common vision held by researchers, technologists, and practitioners has been to seamlessly network live, virtual, and constructive entities into a common environment.

Unfortunately, these environments are notoriously difficult to design, implement, and test due to the concurrency, real-time, and networking characteristics. System designs are complicated by the conflicting requirement to simultaneously connect geographically distributed simulation applications, while each executes and responds to operator and/or hardware inputs in real-time. This conflict can only be resolved by relaxing the consistency of shared data.

We characterize LVCs as a set of asynchronous simulation applications each serving as both producers and consumers of shared state data. Owing to the asynchronous execution and the non-deterministic characteristics of the interconnecting networks, state data used by a consumer is often inconsistent with the most recent value produced. Owing to this, the consistency requirements of dynamic shared state data must be described in terms of accuracy and timeliness: each mapping to a validity interval for each node in the system.

Temporal consistency theory provides a framework for LVC requirements. We have shown that, in terms of data aging, a LVC system can be viewed as a first-order linear system and the rate at which the consumer uses state data is irrelevant to the aging itself. Owing to this, simple analytic models to estimate data aging based upon system architecture can be derived. We have also provided a useful algorithm to compute, in real-time, the temporal consistency of state data for a LVC in operation.

10. Acknowledgements

The authors gratefully acknowledge William K. McQuay from AFRL Sensors Directorate at Wright-Patterson Air Force Base (AFB) for sponsoring this research. We also thank David P. Gehl from L-3 Communications, Christopher Buell from General Dynamics, and Timothy E. Menke and Randyll L. Levine from Wright-Patterson AFB for their thoughtful comments and inputs. We also thank Tim Pokorny, project manager of the 'poRTIco project', an open-source HLA implementation, for his contribution and input regarding implementation details. The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the US Government.

11. References

[1] Department of Defense. 1998. *Department of Defense Modeling and Simulation (M&S) Glossary*, DoD 5000.59-M.

[2] Law, A.M. and W.D. Kelton. 2000. *Simulation Modeling and Analysis*, 3rd edition, McGraw-Hill, New York.

[3] Ghosh, S. and T.S. Lee. 2000. *Modeling and Asychronous Distributed Simulation—Analyzing Complex Systems*, IEEE Press, Piscataway, NJ.

[4] Singhal, S. and M. Zyda. 1999. *Networked Virtual Environments—Design and Implementation*, Addison-Wesley, Reading, MA.
[5] IEEE. 1998. *Standard for Distributed Interactive Simulation*, IEEE Standard 1278.
[6] Object Management Group (OMG). 2007. *Data Distribution Service (DDS)*.
[7] IEEE. 2000. *Standard for Modeling and Simulation High Level Architecture*, IEEE Standard 1516.
[8] Department of Defense. 2002. *The Test and Training Enabling Architecture—Architecture Reference Document*, DoD Foundation Initiative 2010.
[9] Rao, D.M., D.D. Hodson, M. Stieger Jr, C.B. Johnson, P. Kidambi and S. Narayanan. 2009. *Design & Implementation of Virtual and Constructive Simulations Using OpenEaagles*. Linus Publications.
[10] Hamza-lup, F.G. 2004. *Dynamic Shared State Maintenance in Distributed Virtual Environments*. PhD thesis, University of Central Florida.
[11] Kao, B., K.-Y. Lam, B. Adelberg, R. Cheng and T. Lee. 2003. Maintaining temporal consistency of discrete objects in soft real-time database systems. *IEEE Transactions on Computers*, 52: 373–389.
[12] Krishna, C.M. and K.G. Shin. 1997. *Real-time Databases*. McGraw-Hill, New York.
[13] Ramamritham, K. 1993. Real-time databases. *Distributed and Parallel Databases*, 1: 199–226.
[14] Xiong, M., B. Liang, K.-Y. Lam and Y. Guo. 2006. Quality of service guarantee for temporal consistency of real-time transactions. *IEEE Transactions on Knowledge and Data Engineering*, 18(8): 1097–1110.
[15] Gerber, R., S. Hong and M. Saksena. 1995. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transactions on Software Engineering*, 21(7): 579–592.
[16] Ramamritham, K. 2007. Taming the dynamics of distributed data. In *ACM Proceedings of the Eighteenth Conference on Australasian Database*, 2007.
[17] Jensen, K. 1997. *Coloured Petri Nets–Basic Concepts, Analysis Methods and Practical Use*, 2nd edition, Vol. 1–3. Springer, Berlin.
[18] IEEE. 1995. *Standard for Distributed Interactive Simulation—Communication Services and Profiles*, IEEE Standard 1278.2.

Douglas D. Hodson *is a Systems Analyst at the Simulation and Analysis Facility, Wright-Patterson Air Force Base, Ohio. He has 20 years of experience in the domain of modeling and simulation and is the technical lead for the Extensible Architecture for Analysis and Generation of Linked Simulations (*EAAGLES*) software framework. He is also the project leader for the open-source* OPENEAAGLES *development. He received a BSc in Physics from Wright State University in 1985, and both an MSc in Electro-Optics in 1987 and an MBA in 1999 from the University of Dayton. He is currently a PhD Candidate in Computer Engineering at the Air Force Institute of Technology. His research topic characterizes distributed virtual simulations in terms of a temporal consistency model to estimate performance. He is also a DAGSI scholar and a member of Tau Beta Pi.*

Rusty O. Baldwin *is an Associate Professor of Computer Engineering in the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH. He received a BSc in Electrical Engineering (cum laude) from New Mexico State University in 1987, an MSc in Computer Engineering from the Air Force Institute of Technology in 1992, and a PhD in Electrical Engineering from Virginia Polytechnic Institute and State University in 1999. He served 23 years in the United States Air Force. He is a registered Professional Engineer in Ohio and a member of Eta Kappa Nu, and a Senior Member of IEEE. His research interests include computer communication networks, embedded and wireless networking, network warfare, and reconfigurable computing systems.*