View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Calhoun, Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

Theses and Dissertations Thesis Collection

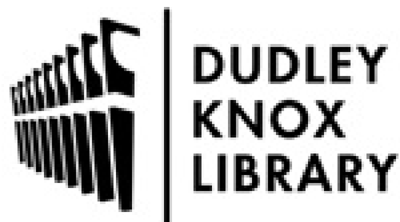
2010-12

A study on Discrete Event Simulation (DES) in a High-Level Architecture (HLA) networked simulation

Wong, Chee Tzuon.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/4958



Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

http://www.nps.edu/library

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

This thesis was performed at the MOVES Institute

A STUDY ON DISCRETE EVENT SIMULATION (DES) IN A HIGH-LEVEL ARCHITECTURE (HLA) NETWORKED SIMULATION

by

Chee Tzuon Wong

December 2010

Thesis Advisor: Arnold Buss
Thesis Co-Advisor: Donald McGregor

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2010	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE A Study on Discrete Event Simulation (DES) in a High-Level Architecture (HLA) Networked Simulation			**5. FUNDING NUMBERS**
6. AUTHOR(S) Chee Tzuon Wong			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Singapore Technologies Electronics (Training & Simulation System) Pte Ltd 24 Ang Mo Kio St 65 Singapore 569061			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** N/A
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ______N.A.__________.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis investigates implementing Discrete Event Simulation (DES) concepts using Simkit packages into a High-Level Architecture (HLA)-networked simulation, thus addressing sustainability of the HLA methodology into the future. Through the DES concept of predicting and anticipating the time of when events will occur, redundant and excessive exchange of common data, like position and sensory status, can be removed. This DES implementation considerably reduces the network load and removes data processing incompatibility between simulations.

A design involving several concepts of DES and HLA simulation led to the creation of a Simkit based application library. Interfacing this application library with two DES models demonstrated and proved the feasibility of DES concepts in HLA-networked simulations. A generic combat scenario modeled using this methodology, successfully showed the intended advantages of the thesis. The ease of linking non-DES and non-HLA simulations to an HLA environment was enhanced using a common set of interfaces built based on the resulting application library. Through a simple comparison with traditional time-stepped real-time simulation of the same scenario configuration, it was shown that data exchange between simulations was reduced by several orders of magnitude. This freed a substantial amount of network resources to perform other tasks, hence, improving network performance.

14. SUBJECT TERMS Discrete Event Simulation, High-Level Architecture, Simkit, Dead Reckoning			**15. NUMBER OF PAGES** 95
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

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

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A STUDY ON DISCRETE EVENT SIMULATION (DES) IN A HIGH-LEVEL ARCHITECTURE (HLA) NETWORKED SIMULATION

Chee Tzuon Wong
Civilian, Singapore Technologies Electronics (Training & Simulation Systems)
B.Eng., Nanyang Technological University, 2004

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN
MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION (MOVES)**

from the

**NAVAL POSTGRADUATE SCHOOL
December 2010**

Author: Chee Tzuon Wong

Approved by: Arnold Buss
Thesis Advisor

Donald McGregor
Thesis Co-Advisor

Mathias Kölsch
Chairman, MOVES Academic Committee

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis investigates implementing Discrete Event Simulation (DES) concepts using Simkit packages into a High-Level Architecture (HLA)-networked simulation, thus addressing sustainability of the HLA methodology into the future. Through the DES concept of predicting and anticipating the time of when events will occur, redundant and excessive exchange of common data, like position and sensory status, can be removed. This DES implementation considerably reduces the network load and removes data processing incompatibility between simulations.

A design involving several concepts of DES and HLA simulation led to the creation of a Simkit based application library. Interfacing this application library with two DES models demonstrated and proved the feasibility of DES concepts in HLA-networked simulations. A generic combat scenario modeled using this methodology, successfully showed the intended advantages of the thesis. The ease of linking non-DES and non-HLA simulations to an HLA environment was enhanced using a common set of interfaces built based on the resulting application library. Through a simple comparison with traditional time-stepped real-time simulation of the same scenario configuration, it was shown that data exchange between simulations was reduced by several orders of magnitude. This freed a substantial amount of network resources to perform other tasks, hence, improving network performance.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
- **A. OVERVIEW 1**
- **B. OBJECTIVES 2**
  - **1. Excessive Data Exchange 3**
  - **2. Network Latency 3**
  - **3. Simulation Synchronization 3**
- **C. THESIS ORGANIZATION 4**

II. BACKGROUND 5
- **A. DISCRETE EVENT SIMULATION 5**
  - **1. Next Event Algorithm 5**
  - **2. Equation of Motion 6**
  - **3. Detection Model 8**
- **B. HIGH-LEVEL ARCHITECTURE 10**
  - **1. HLA Concept 10**
  - **2. Declaration and Management Object Models 12**
  - **3. Time Management 14**
    - ***a. Transportation Services 14***
    - ***b. Time Advancing Services 15***
- **C. SIMKIT 18**
  - **1. Event Graph Methodology 18**
  - **2. Basic Linear Mover 20**

III. ARCHITECTURE DESIGN 23
- **A. NONHLA-COMPLIANT SIMULATORS 23**
  - **1. Data Exchange Reduction 25**
  - **2. Network Latency 26**
  - **3. Synchronized Simulation 26**
- **B. NONHLA-COMPLIANT AND HLA-NETWORKED SIMULATORS 28**

IV. TIME MANAGEMENT DESIGN 29
- **A. TIME MANAGEMENT IN DES 29**
- **B. TIME OFFSET MECHANISM 32**

V. APPLICATION LIBRARY DESIGN 33
- **A. HLA ENVIRONMENT DEFINITION 33**
- **B. EVENT GRAPH COMPONENTS 34**
  - **1. HLA Connection Manager 34**
  - **2. HLA Data Manager 35**
  - **3. HLA Entity List Manager 37**
  - **4. HLA Time Manager 38**
- **C. SUB-CLASSED COMPONENTS 41**

VI. INTERFACING APPLICATIONS 43
- **A. HLA FEDERATE AMBASSADOR 43**

B. HLA DATA ENCODER HELPER 44
C. HLA SIMKIT API 44
1. HLA Environment Setup 46
2. Local Entity Management 47
3. HLA Entity Management 48
4. Time Management 49
VII. TEST AND EVALUATION 51
A. SIMULATION ENVIRONMENT 51
1. Real-Time Platform Reference FOM 51
2. Run-Time Infrastructure 53
3. Simple Movement Detection Simulation 54
B. SCENARIO 55
C. IMPLEMENTATION 56
1. Interface Implementation 57
2. Message Exchange Walkthrough 59
D. RESULTS 63
VIII. CONCLUSION 67
APPENDIX A. HLA FEDERATE AMBASSADOR CALLBACKS 69
APPENDIX B. JAVA HLA SIMKIT FOM OBJECT CLASS 71
APPENDIX C. RPR FOM OBJECT CLASS STRUCTURE [22] 73
LIST OF REFERENCES 75
INITIAL DISTRIBUTION LIST 79

LIST OF FIGURES

Figure 1. Next Event Algorithm (From [4]) 6
Figure 2. Equation of motion for computation of location (X_t, Y_t, Z_t) 7
Figure 3. Basic movement and Cookie-Cutter Sensor detection model (After [5]) 8
Figure 4. Detection equation (From [5]) 9
Figure 5. Federations in execution (From [6]) 11
Figure 6. Communication channels between the Federate and the RTI (After [6]) 12
Figure 7. Registration and discovery process of entity object 13
Figure 8. Two-axis diagram of TSO Events (From [13]) 16
Figure 9. Event-driven Federate using TSO time management (From [12]) 17
Figure 10. Common event graph transition with *t* delay and condition C (After [14]) 18
Figure 11. Simkit java coding convention example 19
Figure 12. Event listener mechanism (After [4]) 19
Figure 13. Event adapter mechanism (After [4]) 20
Figure 14. Basic Linear Mover component event graph (From [5]) 21
Figure 15. Use of application library for interfacing two nonHLA simulations 24
Figure 16. A simulated entity moving in a 5 mile path at 50 mph 25
Figure 17. Data exchange characteristics and rate of update 27
Figure 18. Implementation of one-sided HLA gateway for nonHLA SimEngine 28
Figure 19. TSO message exchange between two event-driven Federates 31
Figure 20. HLA Connection Manager event graph 35
Figure 21. HLA Data Manager event graph 36
Figure 22. HLA Entity List Manager event graph 38
Figure 23. HLA Time Manager event graph 40
Figure 24. HLA Simkit API class event listening mapping 45
Figure 25. Environment setup interface through event listening 46
Figure 26. Local entity management interfaces through event listening 47
Figure 27. HLA entity management interfaces through event listening 49
Figure 28. Time management interface through HLA event listening 50
Figure 29. Base Entity object class structure 53
Figure 30. Simple combat scenario involving a Bomber and Patrolling Aircraft 55
Figure 31. NonHLA Simulation implementation with SMD model 58
Figure 32. UML diagram of function calls for HLA connection setup 60
Figure 33. UML diagram of Federation time synchronization process 61
Figure 34. Simulation walkthrough of messages exchanged 62
Figure 35. SMD simulation display of the synchronized simulation 64
Figure 36. Wireshark records of messages and comparison of results 65

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

DES	Discrete Event Simulation
DIS	Distributed Interactive Simulation
HLA	High-Level Architecture
FOM	Federation Object Model
SOM	Simulation Object Model
DMSO	Defense Modeling and Simulation Office
DoD	Department of Defense
HLA-TM	HLA Time Management
FEL	Future Event List
SimEngine	Simulation Engine
RTI	Run-Time Infrastructure
MOM	Management of Object Model
API	Application Programming Interface
TSO	Time Stamp Ordered
FIFO	First In First Out
LRC	Local RTI Component
SISO	Simulation Interoperability Standards Organization
DLC	Dynamic Link Compatible
JLC	Java Linked-Compatible
RPR FOM	Real-Time Platform Reference FOM
PDU	Protocol Data Unit

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author wishes to express his deepest gratitude to Professor Arnold H. Buss for his patience and guidance during the course of this thesis study. His insightful vision and unreserved imparting of the knowledge in Discrete Event Simulation inspired the author's impetus to research into the methodology of this study. As the creator of the Simkit simulation tool, he provided all possible assistance to the author in the usage and integration of Simkit into the study.

The author also wishes to acknowledge the experience and guidance of his Co-Advisor, Mr. Don McGregor. His extensive knowledge in the field of High-Level Architecture and networked simulation provide a firm foundation and confidence that assisted the author in his design and implementation of the architecture involved.

In addition, the author would like to thank Miss Joy Newman for her expert editing, which molds this study into a professionally written thesis.

The author would like to thank Mr. Koh Kim Leng from Singapore Technologies Electronics (Training and Simulation Systems) for his continuous support and assistance in obtaining a licensed RTI version from MAK technologies.

Lastly, the author wants to express his appreciation to Singapore Technologies Electronics (Training and Simulation Systems) for their sponsorship that gave the author the opportunity to venture further into the arena of Modeling and Simulation and broaden his academic spectrum.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OVERVIEW

Since the evolution of networked simulation, performing joint simulation training involving multiple virtual environment simulators has been feasible. Major players of the simulation industry have been researching and providing multiple network solutions to enable networked simulation training. These solutions created a hidden problem of interoperability identified in the early 1980s by the software standards community. The community believed that, as networked simulation technology matured and stabilized; there would be a need for a common standardization of data exchange across different simulation products from different industries and companies. This resulted in the creation of Distributed Interactive Simulation (DIS) in the late 1980s and, subsequently, High-Level Architecture (HLA) in the mid-1990s.

Although HLA has proven successful in many simulator implementations and designs, network performance, minimizing the amount of data exchanged and time synchronization are still topics of discussion and exploration. Improving network performance and reducing data exchange while maintaining equivalent, if not better, efficiency of a networked simulator has never diminished. As the size of networked simulators grows and data exchanged increases, network performance and reduction of data exchange have become crucial issues.

In the current simulation industry, DIS and HLA are the common architectures used to enable interoperability and networking of multiple simulators for joint and cross-domain simulation training. Neither DIS nor HLA have been able to fully replace customized data exchange format and architecture as the only method for networked simulation data exchange. However there is a trend of increasing utilization of these standards, especially HLA. The requirement for HLA compliance in new military simulation products has been in part responsible for this tendency. With new simulators becoming HLA-compliant, obsolescence of existing simulators and proprietary non-HLA-compliant simulators have created a dilemma; in deciding between disposing of

these simulators versus incurring developmental cost to upgrade them to be HLA enabled. In addition, some simulator developers have used different semantic and syntactic interpretations of the data fields in the standardized Object Model Template (OMT) and customized the Federation Object Model (FOM) in their HLA. This poses major integration and standards compliance difficulties when performing networked simulation between these simulator systems.

Several studies [1], [2] have explored leveraging the characteristics of event agents and discrete events in simulation to reduce excessive data exchange in time-stepped simulations. These studies aimed to reduce semantic and syntactic error in data exchange and reduce the volume data exchanged while retaining simulation fidelity. The usage of Discrete Event Simulation (DES), in particular Simkit, in HLA-networked simulation has not been explored extensively.

Dead reckoning algorithm has been implemented, traditionally, to overcome occurrence of update messages lost due to network latency or unreliable network transport protocol. The dead reckoning algorithm extrapolates or interpolates update points using last know data to justify these lost of data. They, however, have not been implemented extensively to a greater extent to provide higher fidelity and achieve better network bandwidth utilization.

B. OBJECTIVES

This thesis study was motivated by the vision of these problems escalating the increased possibility of crucial integration and implementation difficulties in HLA-networked simulation. The main purpose of this study is to evaluate and to implement the concepts of DES into an HLA-networked environment with the use of Simkit [4], [15]. A Simkit application library for enabling HLA compliance and network data exchange reduction are the targeted results of this study. The thesis study also aims to address several of the emerging problems in the growing trend of HLA-networked environment simulation.

1. Excessive Data Exchange

A characteristic behavior of time step simulation used in conventional HLA simulation is the constant update rate of simulation information onto the network. The amount of data exchanged could be trivial when the number of entities or components simulated is small. More simulated entities and simulator participants in the HLA network environment increases the amount of data exchanged. This can eventually result in degradation of data processing performance in participating simulators [25] and increasing data exchange error.

2. Network Latency

The volume of data exchanged and physical distance between simulator participants is a crucial factor to networked simulation performance. If timely data is important in maintaining a synchronized and smooth simulation display or representation, increasing network latency and delay due to heavy network traffic and propagation delay caused by greater distance is a destructive factor in the networked simulation.

3. Simulation Synchronization

A time-synchronized distributed simulation must maintain a shared simulation clock time; this is a difficult task when there are many simulation participants. This is especially obvious when each simulator has different data processing power and simulation update frequency. This often results in jittering in displaying simulated entities and inaccurate combat results, when data updates are not received in a timely manner and dead reckoning algorithm are not implemented to handle this lapse. In the event that conventional dead reckoning algorithms are implemented, conservative and incorrect implementation still poses difficulties during integration effort.

The above problems prove to be major obstacles in HLA-networked simulation that cause obsolesce of older simulators and reduce the flexibility of HLA network simulation and training. These problems have also been the biggest obstacle when networking simulators using HLA. The resulting Simkit application library of this thesis

study intends to resolve, if not improve, the problem of excessive data exchange, increasing network latency, and synchronization issues through the usage of DES concepts in dead reckoning algorithm and event-driven simulation. An encapsulated implementation of HLA rules, DES as a replacement of conventional dead reckoning algorithms, and a series of common interfaces aims to standardize HLA compliancy and data interpretation without the need to understand these rules or algorithms in details.

C. THESIS ORGANIZATION

The thesis provides, in Chapter I, an overview of the emerging problems of HLA network simulation in the simulation industry and the objectives of this thesis. The basic understandings of the concepts of HLA, DES, and Simkit are illustrated in Chapter II. Subsequently, Chapter III shall explain in detail the design and methodology of the simulation architecture to be used in conjunction with existing simulation products. Chapter IV lists the components of the Simkit application library that caters to common simulation features and requirements. The suggested customization to HLA FOM and possible rule changes will be described in more details in Chapter V. Lastly, Chapters VI and VII detail the test and analysis of the improvements achieved using the implementation and recommended future improvements, respectively.

II. BACKGROUND

Understanding the basic concepts and methodology of Discrete Event Simulation (DES) and High-Level Architecture (HLA), which form the fundamentals of this thesis, are required before performing any design. This chapter will explain the characteristics and methodologies used in the design of the Simkit application library. These methodologies mainly focus on enabling HLA compliancy in Simkit and reducing data interaction within the domain of movement and detecting sensor information.

A. DISCRETE EVENT SIMULATION

Discrete Event Simulation describes an event-oriented methodology of simulation where events may happen at any time. The operation of the system is represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state in the system [3]. During this instant of time, processes involved in the event execution are performed and resulting events from these executions are also scheduled into the event list.

With this characteristic, logical time of simulation is being advanced in an uneven manner instead of the usual regular time duration in a time step simulation. This next point of state change, commonly termed as Next Event, is referenced from an event list. The main purpose of this event list, Future Event List (FEL), is to hold and manage pending events, where future scheduled events are ordered chronologically according to time occurrence [4]. This provides a good indication of length in time that a simulation can be advanced safely to the next point of change. Since events are scheduled into the future and time advancement is discrete and immediate, this provides an "as-fast-as-possible" manner of simulation execution.

1. Next Event Algorithm

To effectively create and design DES models, understanding the next event algorithm is critical. Implementing this mechanism properly would allow the proper

management of scheduled events and advancing time. Figure 1 shows the implementation logic of this algorithm in the form of a state chart diagram [4].

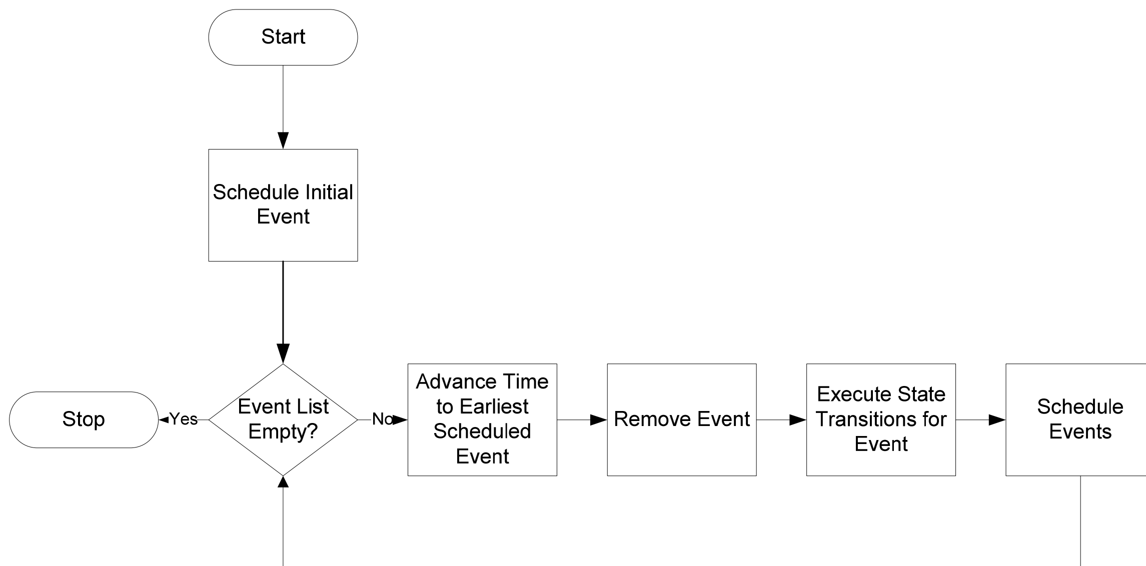


Figure 1. Next Event Algorithm (From [4])

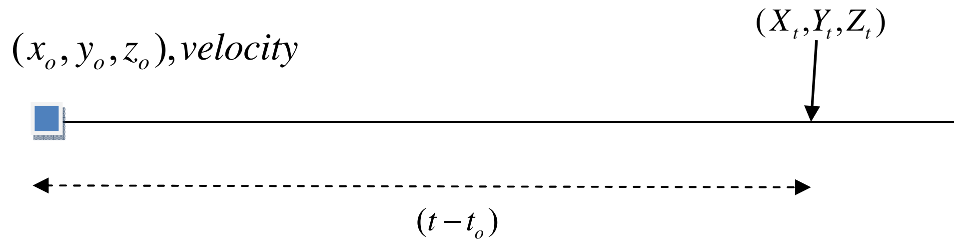
The sequence of activities for the next event algorithm begins with the scheduling of initial events to trigger the initialization of all related variables, parameters, and scheduling of subsequent events. This leads to the start of the iterative process with the check of the FEL. The decision to either proceed to stop the simulation or proceed to jump to the next scheduled event time is based on the availability of next scheduled events on the FEL. Upon advancing simulation time to the next earliest event time instance, the events are removed from the FEL and execution of the state transition is performed. Finally, new events forecasted are inserted into the FEL and events that turn invalid are removed from the FEL. The main characteristic of this event algorithm with the use of the FEL consists of the flexibility of insertion, removal and maintenance of events in the correct chronological sequence pending checks, and execution.

2. Equation of Motion

The characteristics of DES are the principles of scheduling events and information changes into the future. This enables knowledge of the next time of event or

state change. In conventional time step simulation, information of the location of an object in motion is always changing and in constant update. The simple movement model resolves this complexity through the notion of an implicit state of motion [5]. The implicit state of motion defines a state that is not an instance variable with its information explicitly stored in every turn of update. Relevant information, however, is computed based on an “On-Demand” approach. Although complex equations of motion may be considered, this thesis will only utilize the simplest form which describes linear motion

Motion of an object or entity has a linear behavior regardless of distance. The basic linear movement of the object is uniform and can be described with primary information of moving from a starting location, (x_o, y_o, z_o), towards a direction described by a velocity vector, (v_x, v_y, v_z). Hence, the position of the object in motion till time interval, $(t - t_o)$, can easily be computed by applying the equation of motion and determining the distance travelled during this time interval.



$$X_t = x_o + v_x(t - t_o)$$
$$Y_t = y_o + v_y(t - t_o)$$
$$Z_t = z_o + v_z(t - t_o)$$

Figure 2. Equation of motion for computation of location (X_t, Y_t, Z_t)

This model depicts the simple relationship of distance, velocity, and time, which are the main components of motion. Utilizing the same equation of motion, however,

modeling of the acceleration component can be easily incorporated as well through modeling the change behavior of velocity and direction. It will not affect the main concept of compute "On Demand," which affects velocity. This can be forecasted with this dead reckoning algorithm with respect to time.

3. Detection Model

When simulating combat models, besides modeling the movement of entities and objects, sensor detection modeling is the next required aspect of simulation. The simple detection model [5] that this thesis uses depicts several important points of events throughout the whole detection process to describe this sensor detection and entity movement interaction.

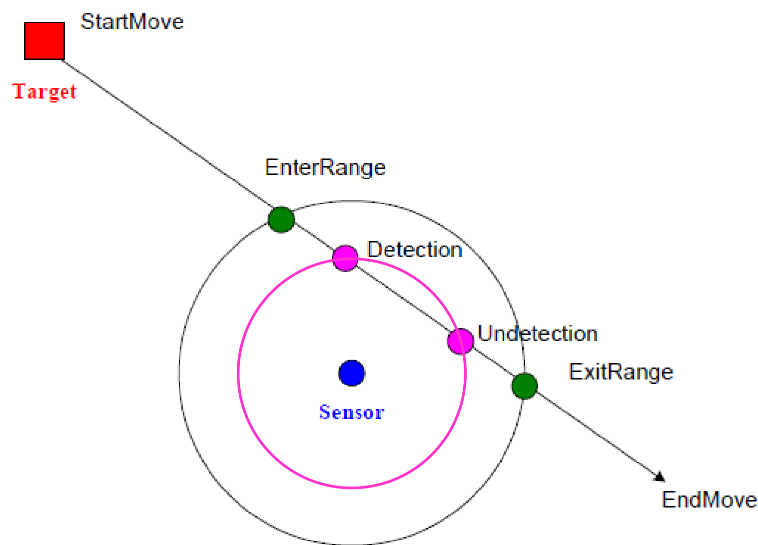


Figure 3. Basic movement and Cookie-Cutter Sensor detection model (After [5])

As a simple example of this movement and detection model, Figure 3 shows the scenario of a stationary sensor representing an air defense sensor with a target moving in a linear path through the sensor detection range. The model classifies the detection into two regions that represent the realistic situation of the air defense sensor detection process. A target begins movement from point of origin and moves into the sensor maximum range of detection. Depending on the sensor capabilities, a confirmed detection

or tracking range is shorter. If no change of action is carried out, there is a high possibility the sensor would lose track of the target and, subsequently, the target would exit the sensor maximum range.

In alignment with DES concepts, these events of movement to enter range, detection, undetection, and eventual exit range, could be computed and scheduled in advance into the event list. These events in turn prove useful to perform other events, such as alert upon entering range, engaging target when detected, reducing weapon effectiveness when tracking is lost, and reducing alert status when target exits range of defense. Although many variations of a scenario exist, these situations are all possible components of a combat operation scenario and can be simulated.

$$t = -\frac{x \cdot v}{\|v\|^2} \pm \frac{\sqrt{\|v\|^2 (R^2 - \|x\|^2) + (x \cdot v)^2}}{\|v\|^2},$$

Figure 4. Detection equation (From [5])

Using the detection equation [5] provides the computation of time, *t,* which includes time to detection, $t_{D,}$ and exit detection, t_E. With the provided start point of target movement, *x,* and movement velocity, *v*, varying the range of the sensor, R, would enable the same calculation at the two regions of range and detection perimeter. Using these timings, the respective events are subsequently scheduled into the FEL during simulation for processing to trigger other state changes.

B. HIGH-LEVEL ARCHITECTURE

The High-Level Architecture (HLA) was developed by the Defense Modeling and Simulation Office (DMSO) of the US Department of Defense (DoD) to meet the needs of defense-related projects [6]. Through its HLA initiative, the DMSO intends to address the continuing need for interoperability between new and existing simulations. DMSO hopes to achieve this by providing a common technical framework and a standardized architecture for interoperability and enhance of the reusability of common modeling and simulation components.

1. HLA Concept

The main difference between DIS and HLA is the implementation of the HLA concept that did not standardize the format in which information is exchanged (as with DIS), but only interfaces and services among simulation applications. The HLA concept consists of three parts. These are a set of HLA simulation rules that govern the characteristics of HLA-compliant simulations, an object modeling scheme, and an interface specification. The set of ten HLA rules indicates the common guidelines that a system has to follow for creation and management of Federation and Federates. Adhering to these rules makes a simulation system HLA-compliant.

The HLA paradigm requires the implementation of the concepts of Federation and Federates. The Federation refers to the overall simulation environment, and its participating members are identified as Federates. These Federates join the Federation to exchange information according to a common Federation Object Model (FOM) which is designed in accordance to the Object Model Template (OMT) defined in the IEEE 1516 standards [7].

The FOM is a consolidated list of types of objects and their attribute values that are exchanged within a Federation. The FOM specifies the objects and attributes that Federates can publish and subscribe and allows data exchange in a controlled and standardized manner. Federates, however, are not required to simulate and provide information for all object formats indicated in the FOM. A Simulation Object Model

(SOM) residing in each individual Federate serves the purpose in specifying the types of information that the individual Federate is interested in receiving and the types of information that it would provide to the Federation.

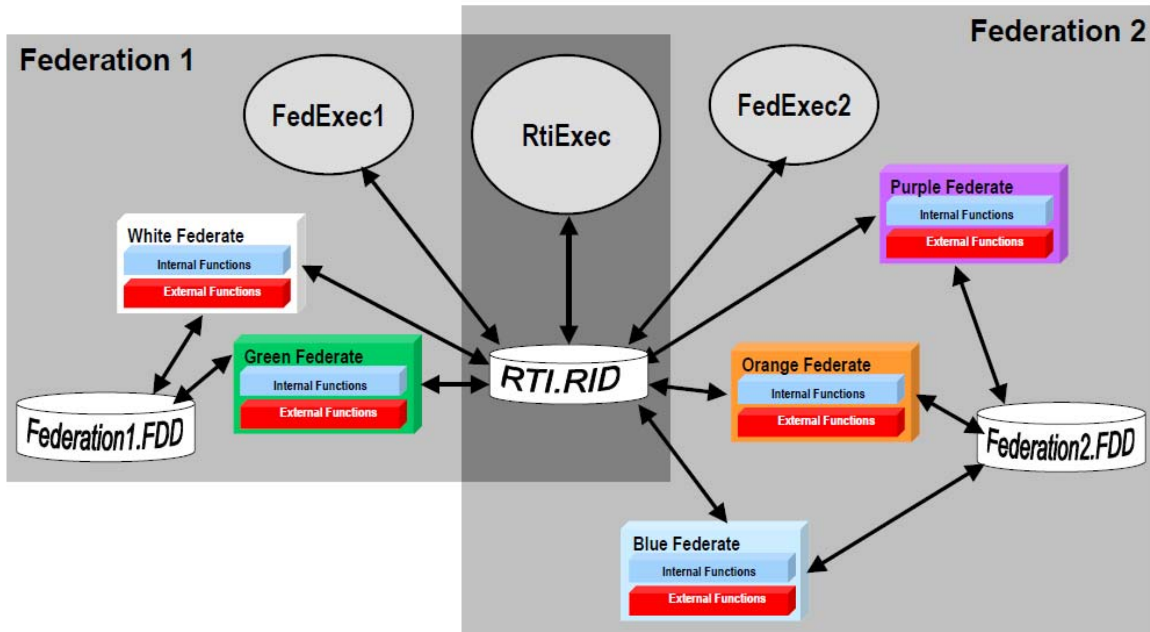


Figure 5. Federations in execution (From [6])

After creating Federations and Federates, an interface is required to provide the medium for data exchange. The Run-Time Infrastructure (RTI) provides the required interface specifications that the software environment needs to exchange information in a coordinated fashion [8]. Federates in the Federation must communicate with each other via the RTI. Figure 6 shows the basic communication channels between the Federate and the RTI. The RTI ambassador provides the interface for a Federate to send information to the Federation. The implementation of standardized callback services in the Federate ambassador allows the Federate to receive the corresponding information and, subsequently, process the information internally in the Federate.

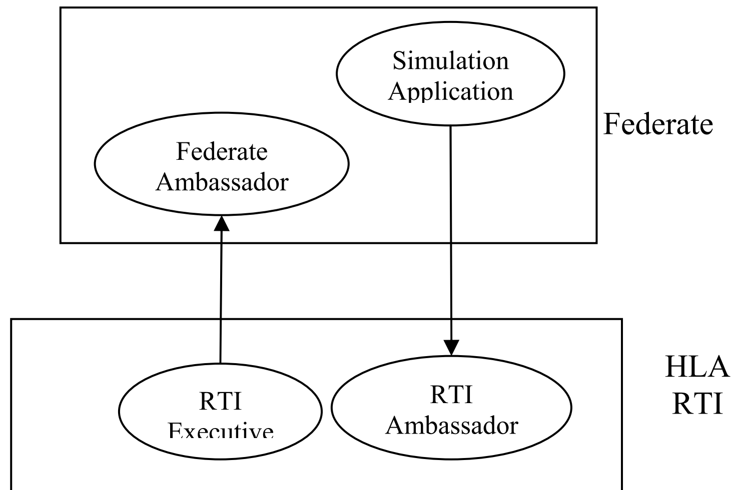


Figure 6. Communication channels between the Federate and the RTI (After [6])

2. Declaration and Management Object Models

The IEEE 1516 standards [7] define several management designs that are required to enable ease of HLA simulation. Two of the primary services that require implementation are the Declaration Management service and the Management Object Models (MOM). The HLA MOM concept implies that a Federation execution can be managed by a combination of Federate and RTI supplied information. Specifically, it consists of a set of predefined object and interaction classes that provides a manager Federate with the capability to monitor and control aspects of the Federation using the standard RTI run-time services [9], [10].

The major concepts involved in the implementation of the MOM service are the common interface of publishing, subscribing and registering of objects, data formats, and interactions. The publishing process declares the data types specified in the SOM of the Federate to the Federation. This informs the RTI of the type of data formats and interactions that the particular Federate is capable of producing and provides updates. On the receiving end of this MOM service is the interface of subscription, where the Federate indicates the data format and interactions that are of interest to the applications within the Federate. This interaction of publish and subscribe provides the baseline and partial controlling factors to entity data exchange.

Upon simulation execution, the declaration of these data types for communication follows the creation, updating, and deletion of objects and interaction. These actions are achieved using the standardized API calls in the RTI and Federation ambassadors. The *registerObjectInstance*() function call announces to the RTI the existence of the entity object within the simulated Federation. At this instance, the RTI would inform all Federates that indicated their interest in this object type during the subscription process, of the creation of this entity object through the callback *discoverObjectInstance()*. This forms a continuous process of registering and discovering of the entity object within the Federation for a common picture of simulation.

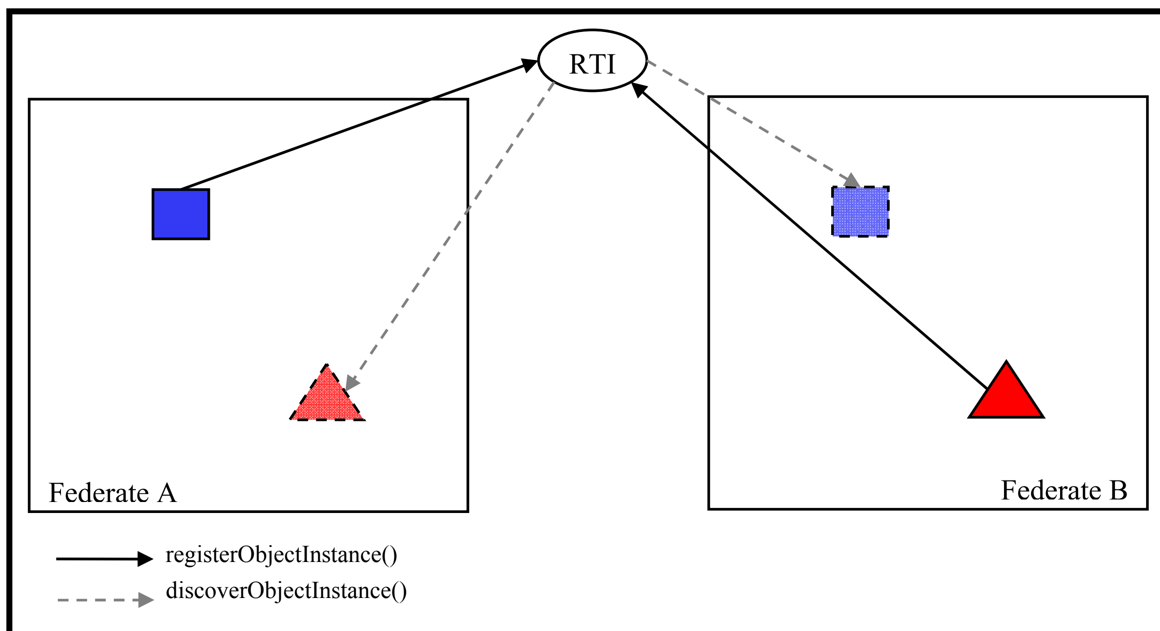


Figure 7. Registration and discovery process of entity object

Attribute changes and updates of these object instances are interfaced with *updateAttributeValues*() for sending updates and *reflectAttributesValues*() for receiving changes. HLA interactions, which are events, have similar interfacing functions *sendInteractions*() and *receiveInteractions*() for sending and receiving, respectively. Lastly, *deleteObjectInstance*() removes the object instances that are invalid or destroyed in the simulation arena. The MOM forms the controlling agent in information exchange for HLA simulation.

3. Time Management

The HLA Time Management (HLA-TM) is concerned with the mechanisms for controlling the advancement of time during the execution of a Federation. Time advancement mechanisms must be coordinated with other mechanisms responsible for delivering information, e.g., to ensure messages are not delivered in the past of the logical time of a Federate [11]. This service has the main purpose to support interoperability among Federates utilizing different internal time management mechanisms [12].

To implement these time management services, two aspects of the Federation execution must be considered: the transport for delivery of messages, and the type of time advance service to be used. The message transport type chosen is based on cost, network performance, and bandwidth consumption characteristics. Time advance mechanism is chosen based on the characteristics of the simulation and will determine the control measures used in Federate time advance.

a. Transportation Services

The different transport services are categorized based on two characteristics: the reliability of message delivery and message ordering. The reliability of message delivery refers to delivery by the RTI through retransmission or best effort delivery. This is a tradeoff between network latency and jitter versus probability of successful delivery of the message. Message ordering in HLA consists of five types of delivery mechanisms: receive, priority, causal, totally ordered, and time stamp ordered. The type of mechanism used in a simulation depends on the type of message required for the simulation execution.

Receive Order is the most direct and lowest latency ordering mechanism. Messages are passed to the Federate in the order that they were received. The incoming messages are placed at the end of a first-in-first-out (FIFO) queue, which, subsequently, is sent to the Federate by removing them from the front of the queue. This message order type is usually used by hard real time simulation [12]. Priority Order stores the messages in a priority queue, where the time stamp denotes the priority of the messages. Messages

with order of the smallest time stamp are sent to Federate first. Thus, this mechanism does not prevent messages from reaching in the 'past' of the Federate. Causal Order is a more complicated mechanism where messages are sent in both order of time and in order of occurrence. For example, if message A is indicated to happen before message B, even when message B is received by the RTI first with a smaller time stamp, the RTI will hold message B in its buffer and wait for an instance of message A before sending message B. This is a more stringent method of implementing message order.

Lastly, the Time Stamp Order (TSO) is the mechanism used commonly in DES. A message sent to the RTI requires a time indication of when the event or update occurs along the logical time of the Federate execution. The RTI will store all of the messages in its buffer and only send the messages when it can be sure that there will be no messages delivered to the RTI subsequently that contain a smaller time stamp order. The RTI ensures this condition through the time advancing service described in the next section. One of the characteristics of the TSO, when handling messages with the same time stamp, is that they would be delivered to the Federate in the same order that they were received. This provides an implicit ordering of messages.

b. Time Advancing Services

Time Advancing services requires an HLA execution to be either time constrained, time regulating, or both. If a Federate is defined to be time constrained, it is able to receive TSO messages and is limited by the time advancement of other Federates. Time regulating characterizes a Federate to be able to send TSO messages and determines the logical time of other Federates. Figure 8 shows the definitions of a regulating Federate and a constrained Federate.

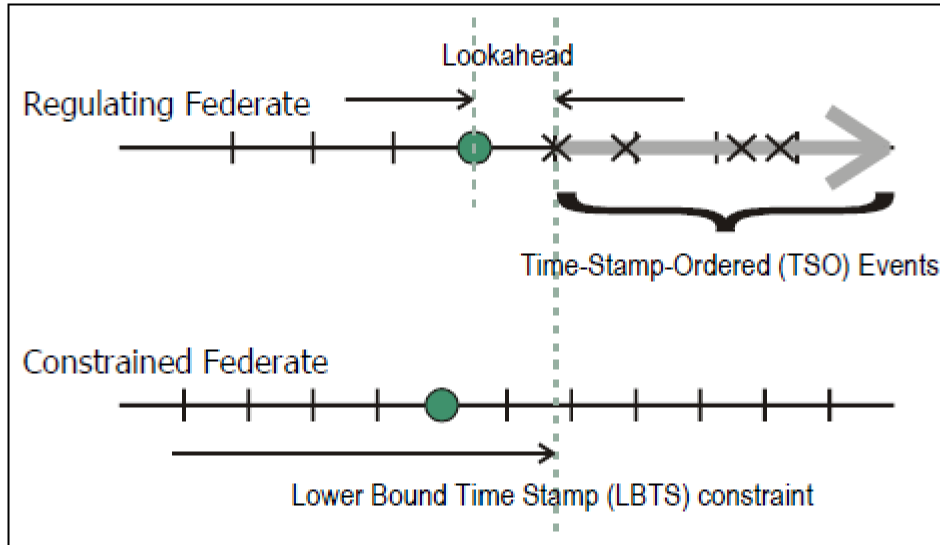


Figure 8. Two-axis diagram of TSO Events (From [13])

A regulating Federate has to deliver TSO messages with a time stamp equal to or larger than the Lower Bound Time Stamp (LBTS). This LBTS typically is the current logical time of the regulating Federate with addition of a Lookahead time. The Lookahead time serves the purpose of ensuring that TSO messages forwarded do not lag behind the logical time of the constrained Federate, i.e., constrained Federate will not receive TSO messages of the 'past.'

To maintain this implementation of time advancing with time regulating and time constrained, the regulating Federate uses the HLA function call, *timeAdvanceRequest()* and *nextEventRequest()* to request for a time advancing grant. Once a Federate evokes either of these two messages, it guarantees that no TSO messages with time stamp less than the LBTS would be sent. The RTI subsequently makes the decision of sending a *timeAdvanceGrant()* message to allow the regulating Federate to advance time to the logical time stamp indicated in the grant. This process of request and grant provides a controlled time advancement environment.

A good example that demonstrates this process is extracted from HLA Time Management Design Document [12]. Using a wall clock time to synchronize the LBTS enables a coordinated time advancing and message exchange process. The

important concept depicted in this example is the interaction of message exchange when the interfaces, *sendInteraction()* and *nextEventRequest()*, are invoked. It shows that after the event-driven Federate processes all local messages and events, it sends interaction updates with time stamp 40 and announces the next local event time with a time stamp of 42.

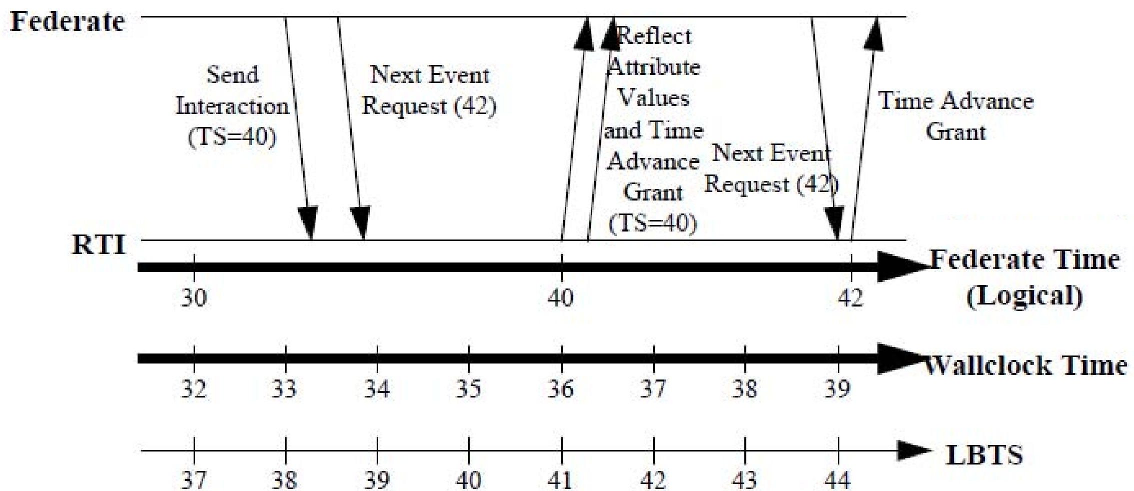


Figure 9. Event-driven Federate using TSO time management (From [12])

The RTI holds all TSO messages in its buffer until the LBTS advances beyond the time stamp of 40. At the point of LBTS at 41, the RTI forwards all relevant messages with time stamp smaller than the LBTS and grants time advancement to logical time 40. A resend of *nextEventRequest()* with time stamp 42 was made to request time advancement to the intended logical time.

C. SIMKIT

Simkit is an application library written in Java that harnesses the methodology of event graphs and discrete event scheduling paradigm. Its main objective is to enable ease of designing and creating discrete event-driven simulation as an open source toolkit. In this section of the thesis, the basic concepts of event graph paradigm, the Basic Linear Mover and Cookie Cutter sensor library classes are explained to provide foundation knowledge on the use of these classes in the design that follows in this thesis.

1. Event Graph Methodology

Event graph methodology is an attempt to use graphical means to explain the states and transitions of a DES model. This graphical representation is simple in nature and its expression strongly reflects the event-driven nature of event-oriented systems. The strength of its simplicity has tremendous value in enabling ease of analysis, especially in perceiving the sophistication of event-scheduling approaches in discrete-event system simulation [14]. A common event graph transition depicted in Figure 10 shows the transition from an event A to an event B on the condition of C. Thus, event B is scheduled into the future of the event list with delay *t* in time. The event graph also provides the information that an argument k is being passed to event B on the scheduling edge of the transition.

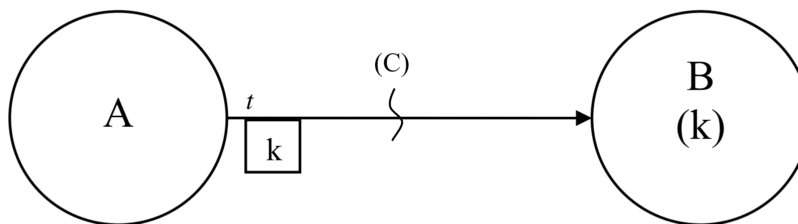


Figure 10. Common event graph transition with *t* delay and condition C (After [14])

The event graph methodology is closely related to coding convention in Simkit. Every component of the overall event graph represents a Java class in the DES model and an event in the component is related to a function call with "do" appended in front of the function name as the usual naming convention. Using Figure 10 as an example event graph will yield a series of java code states as follows.

```
public void doA()
{
   If (C)
            waitDelay("B", t, k);
}
public void doB(k) {…}
```

Figure 11. Simkit java coding convention example

Another feature in the Simkit application library is the event listener and adapter mechanism. This mechanism provides the interface that links two separate components together. Each component in Simkit has an independent set of event-graph logic. The triggering of events within a component can cause dependency by other components in such a way that a system event occurring in a source component triggers the execution of the same kind of event in another dependent or listening component. This is the underlying concept of the listener mechanism in which there is an event-source component, an event-listener component, and a line that connects the two with a stethoscope symbol on the source end [15].

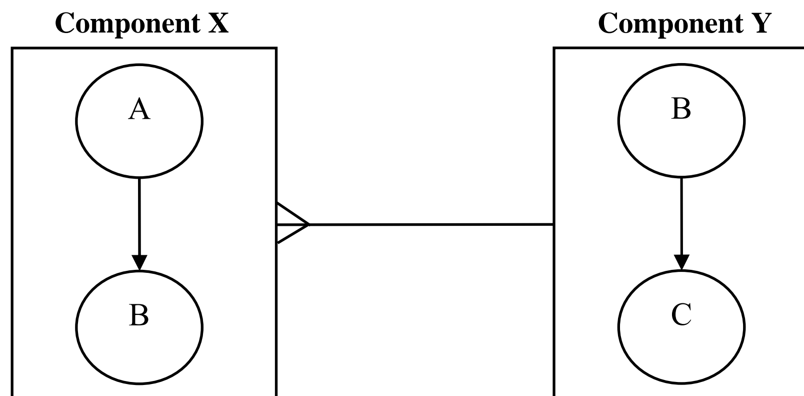


Figure 12. Event listener mechanism (After [4])

The event adapter is the more specific version of the event listener mechanism. The event listener allows the source component to trigger all dependency events in the listening component as long as events are of the same type and format. The event adapter allows a more deliberate listening of specific type of event. An event B in component X would be able to trigger an event C in the component Y as depicted in Figure 13, which shows the event graph annotation of an adaptor interface. With the use of the event adapter, Event B in component Y would not be executed unlike previous case depicted in Figure 12. This provides a more controlled manner of utilization of the listening mechanism to adhere to strict conformity of events. The adapter mechanism is useful when not all event types in the listening component intend to be dependent on the source component for activation.

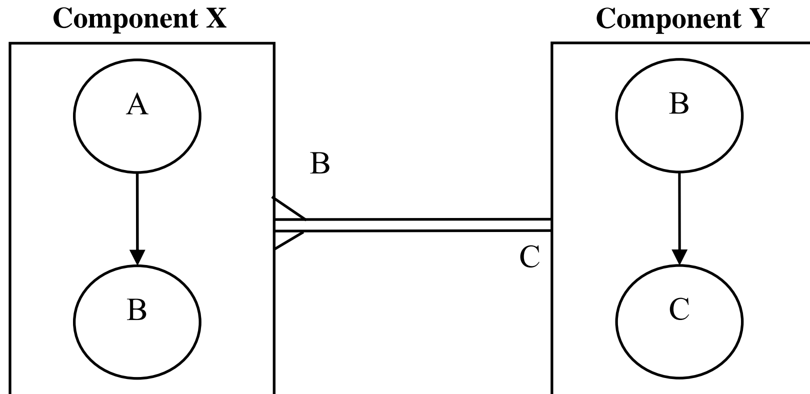


Figure 13. Event adapter mechanism (After [4])

2. Basic Linear Mover

As explained in the sections of equation of motion and detection model in the beginning of this chapter, it is more efficient to model linear and uniform movement of an object through forecasting the ending location of the object into the future. The Simple Movement and Detection (SMD) part of the Simkit library consists of the Basic Linear Mover component, intended to allow easier modeling of the movement of simulated objects by utilizing the equation of motion to describe the future location of the object. The basic linear mover component of Simkit uses a simple event graph to initialize the

start of movement and end of movement. The “Start Move” event indicates the beginning of the movement which records all initial parameters and conditions. The “End Move” event marks the completion of the linear movement and flags awareness in possible changes of parameters in the movement of the object. With starting and ending points, the component determines the required time to travel between the two points. Hence, the end of movement event could be scheduled into the future with the computed movement time, t_M, as delay. Current location of the object along the time of movement can be computed and returned for the purpose of display or other activities.

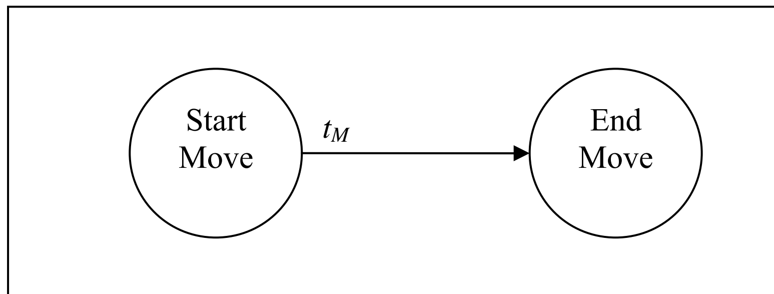


Figure 14. Basic Linear Mover component event graph (From [5])

As an extension in modeling the motion of object with Simkit, the Basic Linear Mover component can be used in conjunction with the various mover managers: Path Mover Manager, Patrol Mover Manager, and Random Mover Manager. These mover managers aim to model movement characteristic of pre-defined path, patrolling movement in a recursive manner, and a randomized destination movement pattern, respectively.

THIS PAGE INTENTIONALLY LEFT BLANK

III. ARCHITECTURE DESIGN

Following an initial detailed research of the basic concepts in High-Level Architecture (HLA) and Discrete Event Simulation (DES), the overall architectural design of the intended resulting application library was researched as to applicability to meet the identified objectives. As a review of the objectives, the resulting design aims to fulfill three objectives: reducing excessive data exchange, decreasing the effects of network latency in simulation, and compatibility adaptation of simulator with different capability to perform synchronized HLA simulation. The aim of having the ability to implement HLA in a fast, simple, and accurate manner, without the need to understand its rules in details, was placed as one of the objectives of the architecture design. This is to relieve simulation developers from a lengthy and tedious process when developing an HLA simulation from the beginning.

Having these objectives in mind, there are a total of two approaches to implement and to use the proposed architecture design of the resulting application library. The application library is designed with the intention to serve as a gateway between the simulators, regardless of HLA compliancy, simulation engine[1] performance or simulation type[2]. The simulation engine (SimEngine) is in charge of implementing the application library and the various interfaces to enable the operability of this architecture.

A. NONHLA-COMPLIANT SIMULATORS

Through a detailed research on the current problems in HLA-networked simulation, the complex interfaces and required standardization of HLA rules deters simulator developers from implementing the HLA standards or implementation with full adherence to the HLA rules. The approach used in this thesis study is designed to enable ease of connecting to a HLA simulation environment for nonHLA-compliant simulators.

[1] The application in a simulator performing computation, interfacing, and management of simulated objects.

[2] Constructive or virtual simulators for air, land or sea domain platforms.

Figure 15 shows that a SimEngine is not required to understand any of the HLA rules or interfaces to the Run-Time Infrastructure (RTI). The main requirements of the nonHLA-compliant simulators are to implement the interfaces of the resulting Simkit application library from this study and to provide a common RTI through installation from any available source. The focus of the interfaces is to provide simple method linkages and allow communication between the SimEngine and the application library. These interfaces will contain generic function calls for creation, updating, and deletion of simulated entities to be sent to the simulator at the other end. Callback functions exist in the interfaces simultaneously to provide a means of receiving data from the HLA environment.

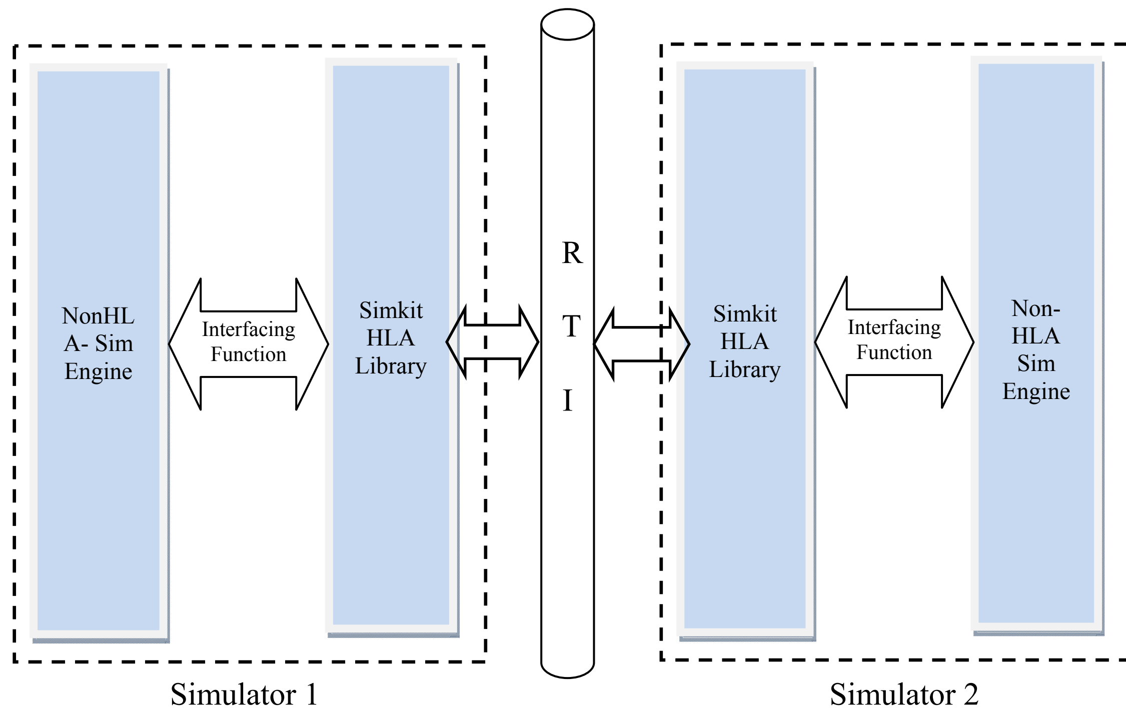


Figure 15. Use of application library for interfacing two nonHLA simulations

This architecture, designed as a method of use of the resulting application, targets to yield several advantages and satisfies most of the objectives in this thesis study.

1. Data Exchange Reduction

To reduce excessive data exchange, use of the DES concept of updating upon change is used in both communication between the SimEngine to Simkit HLA application and bidirectional between the Simkit HLA applications via the RTI. The expected improvement in data exchange reduction can be measured through a simple example of position data of an entity, moving in a two way point path, sent from a 30 updates per sec (30Hz) timed simulation to another simulation in the HLA network.

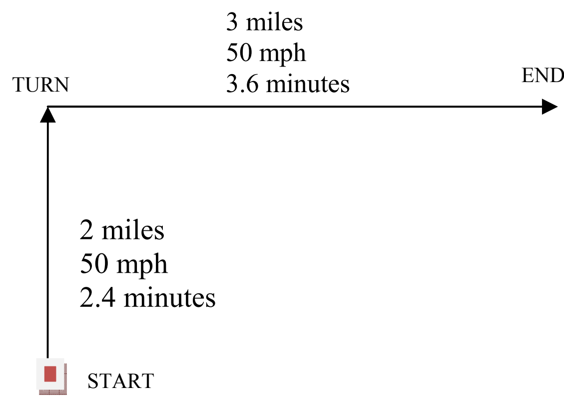


Figure 16. A simulated entity moving in a 5 mile path at 50 mph[3]

If the entity is moving at a speed of 50 miles per hour (mph) on the path as depicted in Figure 16, it would take the entity six minutes to complete a five miles path in real time. When the traditional time-stepped simulation is used for HLA network, there would be 10,800 updates of the entity location sent to the destination simulation for processing. In this Simkit architecture design, there would only be three updates sent, during starting point of the movement, at the turning point where the entity direction is changed, and when the entity stops at the end point of the movement. There is a significant improvement in the amount of data exchange with this architectural design.

[3] An illustration of the reduction of data exchange, this situation might not be realistic in normal combat modeling scenarios.

2. Network Latency

As discussed previously, when the data exchange is reduced significantly, the number of data packets commuting on the network is reduced; thus, the RTI and network devices process less data. The network would have better bandwidth efficiency and availability of processing resources. This is expected to reduce the processing and propagation delays in the overall network latency. This method of data relation, however, causes every data packet exchanged in this architecture design to become extremely critical and sensitive to any amount of network delays or data lost in transfer. A reliable transport protocol is required in this architecture to ensure that the data packets exchanged are always performed successfully.

3. Synchronized Simulation

One of the problems in HLA-networked simulation is synchronizing simulation of different performances. As the simulation executes, there will be some detrimental data updates between simulators performing updates at different rates. Network latency and loss of data packets would further aggravate this causing the problem of jittering or a "teleport" image correlation phenomenon. Time step simulation traditionally implements a dead reckoning algorithm to overcome the problems of data packet loss or to make the display more visually appealing despite using low data update rates.

This proposed approach overcomes these problems with the Simkit application library, which manages the changes of HLA entities[4] data. The SimEngine performs update requests when simulated HLA entities data are required for display. Updating data for local simulated entities is still performed based on the "upon change" concept. Implementing this architecture would separate the required update rate of the SimEngine to display HLA entities from the data update characteristics of local entities by the Simkit application library to the RTI. Regardless of required update rate of display by the SimEngine, the Simkit application will reply with the computed data of the requested

[4] These are entities that belong to other simulators and their data are updated from the RTI.

HLA entity based on the equation of motion and up to date data from the Federation. This resolves the issue of different update rates between HLA-networked Federates that cause correlation problems.

To illustrate this characteristic of the architecture, Figure 17 shows the graphical display of simulator 1 with two entities: local entity A and an HLA entity B. It also includes the data exchange rate characteristics between the SimEngine, Simkit application library, and the HLA Federation. Local entity movement data is updated to the Simkit application library with the "upon change" concept of DES. The Simkit application library, subsequently, updates this information into the Federation. HLA entity is displayed graphically in the simulator with the data requested from the Simkit application library. There is no direct interaction between the SimEngine and the RTI to obtain data of the HLA entity. Rather, it is obtained from the Simkit application library.

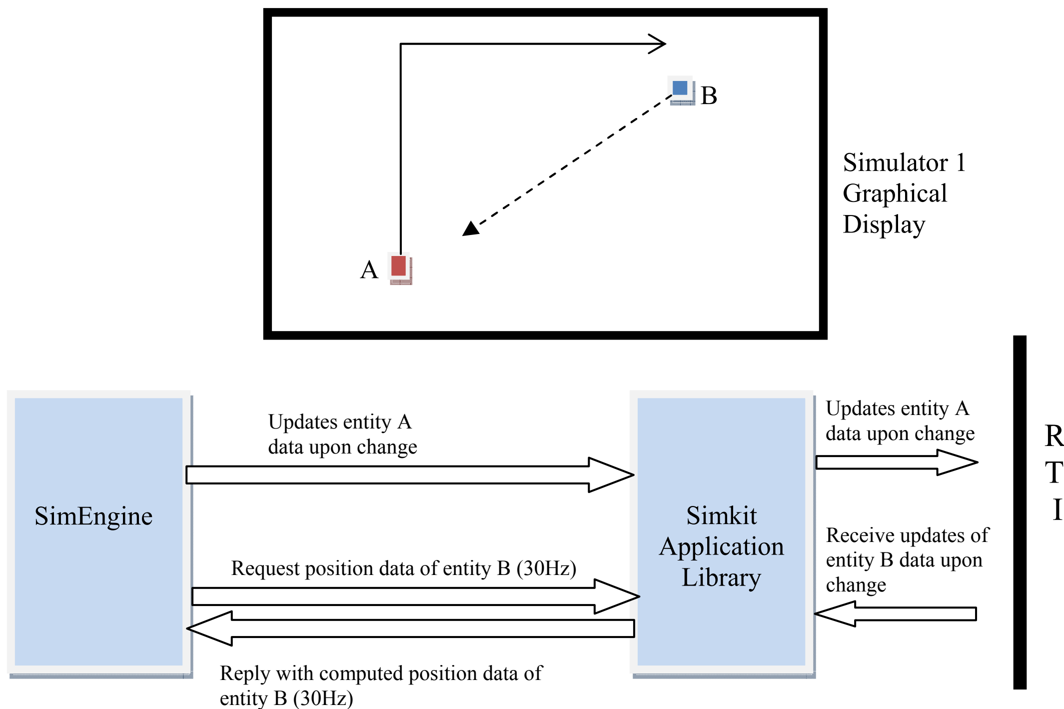


Figure 17. Data exchange characteristics and rate of update

B. NONHLA-COMPLIANT AND HLA-NETWORKED SIMULATORS

The other usage approach for the application library is a one-sided gateway of enabling a nonHLA SimEngine to communicate with an HLA-compliant networked SimEngine. This approach has the same advantages as the previous method of implementation. These advantages, however, depend on the definition of HLA entity data exchange of the HLA-compliant SimEngine. If the HLA-compliant SimEngine defines its data exchange using the DES concept of event-driven data definition, i.e., data represents entity change of state and updates are sent upon change, this approaches yields exactly the same pro factors. In opposite cases, when it is performing data exchange based on the time-stepped simulation definition, where entities' data is sent at each constant time interval, this approach will only be beneficial for ease of HLA interoperability implementation. In the latter assumption, the amount of data exchanged will not be changed and this does not help in improving network performance. A synchronized HLA-networked simulation still can be achieved at the nonHLA SimEngine side. This is because the concept of dead reckoning still exists which will assist in filling the gap when entity data packets are lost or arrive late into the simulation.

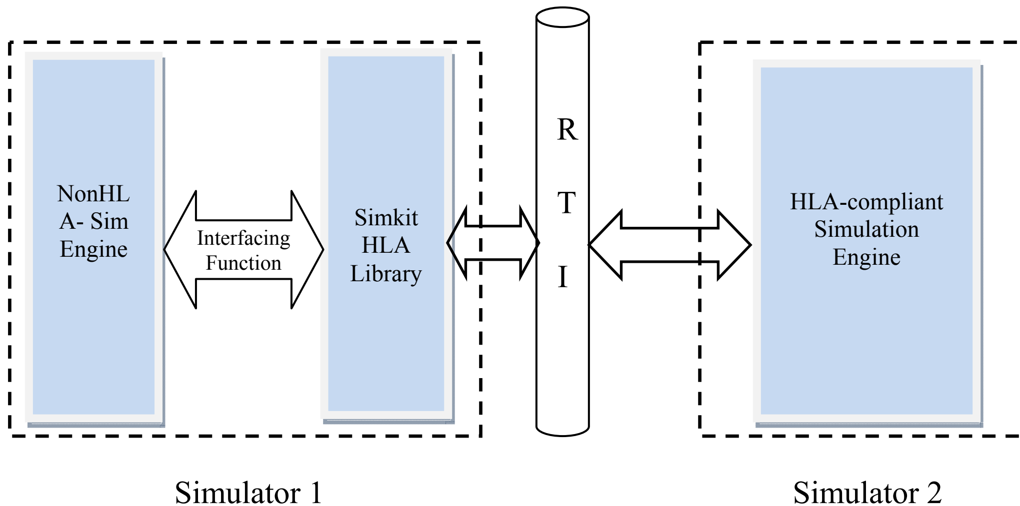


Figure 18. Implementation of one-sided HLA gateway for nonHLA SimEngine

IV. TIME MANAGEMENT DESIGN

When real-time simulation was placed into consideration, the need for some form of time management in the design of the application library was identified. Several previous works with time management [8], [11], [12] and implementation of Discrete Event Simulation (DES) in High-Level Architecture (HLA) [1], [2] was referenced, but there was no significant example or similar work related to the usage of DES to perform real-time simulation in an HLA-networked environment. After placing detailed research into the HLA Time Management (HLA-TM) design [12], a combination of an event-driven simulation and usage of the time information in the Time Stamp Ordered (TSO) messages methodology was adopted. As a review of the HLA-TM implementation explained in Chapter II, this chapter will provide a detailed explanation of the various interfaces in terms of their uses to synchronize time and their roles in a simulation. This consists of two event-driven Federates.

A. TIME MANAGEMENT IN DES

In accordance to the HLA standards, the time management design provides a progressive time advancing methodology. Before explaining the details of this design, there are several terminologies of time that need to be defined: Simulation Time, Federate Time, Lookahead Time, Lower Bound Time Stamp (LBTS), and Wall Clock Time.

The Simulation Time is the time component of the Simkit-modeled DES simulation that denotes the length of time the event list has been executed. All scheduled events in the event list follows this execution time. The Federate Time is the time component which the Local RTI Component (LRC) perceives that the Federate's simulation has proceeded. This is important timing to the RTI host as it marks when the RTI host should be sending the corresponding TSO messages to each Federate.

The Lookahead Time is an interval of time used with the Simulation Time in computing the LBTS. It acts as a safety interval bringing the LBTS far ahead of time, sufficiently to prevent any late arrival of messages into the simulation. The LBTS is the

boundary required in a time regulating HLA simulation. It limits and acts as a declaration to the RTI host the time limitation of time stamp messages that can be sent. This means that each time regulating Federate is not allowed to send any TSO messages with a time stamp less than the LBTS.

Lastly, the wall clock time is an independent time of the system that is not affected by any factors. The wall clock time is determined by the system clock of the hardware platform. To ensure all participants of the networked simulation are running on the same time, this time component is usually synchronized via a time domain server.

After understanding these terminologies, using the event-driven Federate time implementation in HLA-TM design document [12], the design is expanded further into a simple two event-driven Federate's simulation to illustrate this time management design. At the start of the application, both Federates are declared to be time constrained and regulating during the Federate creation. This is a typical setup for an event-driven Federate.

The HLA-TM design of the resulting application library utilizes the *updateAttributeValue ( )* and *nextMessageRequest ( )* to send the entity data updates and declaration of the next time an event will occur. Subsequently, the RTI forwards the updating messages to the destination Federates and answers the time advance request with *reflectAttributeValue( )* and *timeAdvanceGrant( ),* respectively. Figure 19 depicts an example of how the TSO messages are exchanged. It also illustrates the time advancing mechanism in a real time event-driven simulation. In this design, the LBTS is tagged to the simulation time of each Simkit simulation with an addition of the Lookahead time of 1 stated in the example. At the beginning of the simulation, both Federates perform an initial update of data at LBTS equal to 1. The updating message is sent with a time stamp of value equal to the LBTS. A request to advance time to the next event is made, subsequently, to declare that there are no other TSO messages with time stamps less than the next event time. In the example, the time stamp of the first next event request message of the Federates A and Federates B are 5 and 3, respectively. Upon receiving these two requests, the RTI would send the updates to the destination Federates with time stamp of 1. It also grants the time advancement to the lower time stamp value Federate,

which is Federate B of time stamp 3. This decision of which time advancing request to grant is decided by the RTI based on the smallest time interval advance. It will not send any messages or time advance grants unless it can be sure that no Federates will send any TSO message with less than the nearest requested time interval.

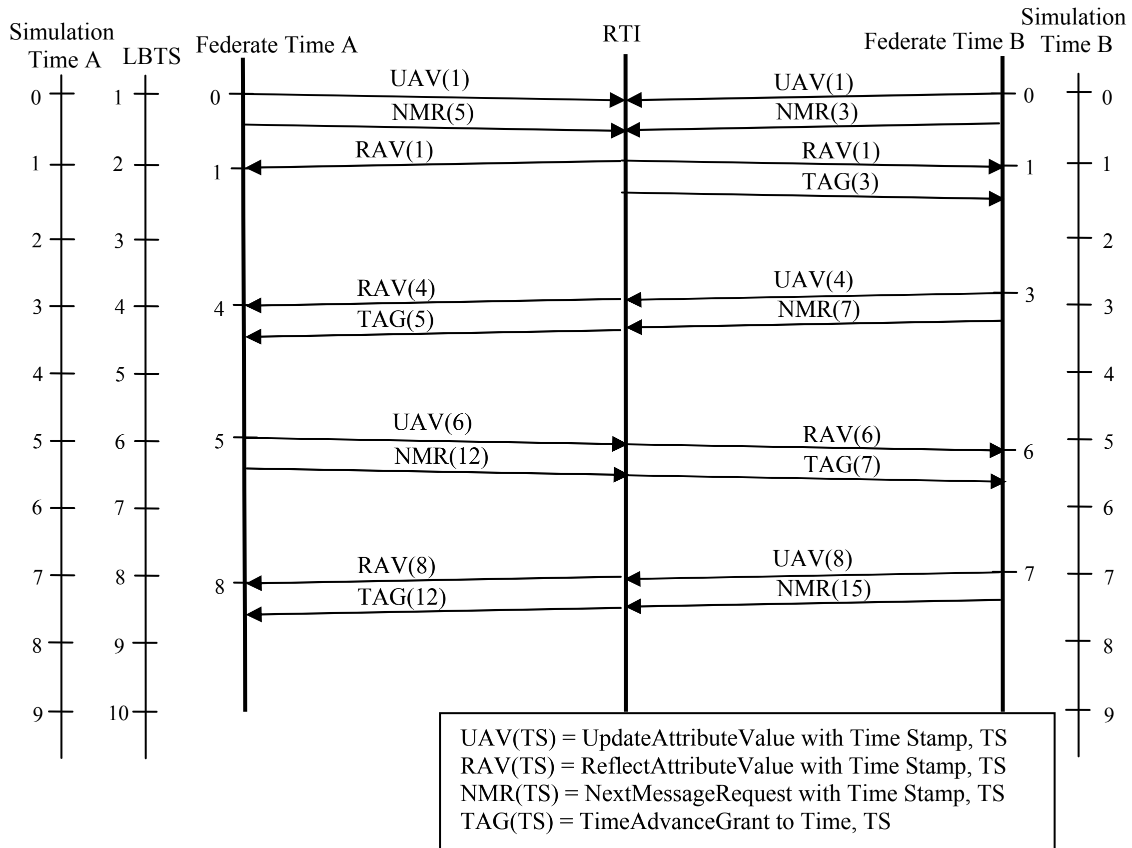


Figure 19. TSO message exchange between two event-driven Federates

In the event that the TSO messages contain a time stamp that is substantially into the future, the messages are stored in the RTI until the LBTS has proceeded greater than time stamp of the received TSO message. This is a cyclic process of Federate sending updated TSO messages when its entities have changed properties. It requests time advancement according to the next possible time of event occurrence. The RTI holds onto TSO messages and forwards them to destination Federates only when their time stamps have passed, and grants the time advance to the Federate that has the request of the smallest interval jump in time. If the simulation proceeds, this process is performed

through the example at the time 0, 1, 3, 5, 7, and 12. To assist in maintaining a synchronized time advance between the Federates, a small modification was made to the original HLA-TM design. This modification uses the time stamp of the forwarded TSO messages to advance the Federates time besides the time advance grant reply. This is to allow Federates to advance in time simultaneously and not in a racing manner.

B. TIME OFFSET MECHANISM

Although the time management design was modified to maintain synchronization in the time advancing of the Federates, there still exists a small amount of time racing. This is especially prominent when a large number of entities or objects are involved in the simulation. When there is a need to process many messages simultaneously during real-time simulation, the time required to process these messages will result in a small delay to advance in time. A "roll back" mechanism is usually used to resolve this in DES or event-driven simulation, but in this design a waiting algorithm was adopted.

A limiting value is set in each of the Federates to trigger the time offset mechanism. Using the difference between Simkit simulation time and the time stamp of the last TSO message received, the amount of time offset to execute is determined. When this difference is larger than the stated limiting value, the offset mechanism will be activated. Before resuming, the Federate leading in the simulation will go into a wait state and pauses all simulation for the amount of time difference.

V. APPLICATION LIBRARY DESIGN

This chapter gives a detailed explanation of the Simkit High-Level Architecture (HLA) application library design. The design and implementation of this application library is done using the event graph methodology. The components are designed with the purpose of enabling Simkit to be HLA-compliant and to maintain the entities of the simulation. Some sub-classes of the Simkit components were also designed with the aim to support the management of the Discrete Event Simulation (DES) concepts of event and motion prediction.

A. HLA ENVIRONMENT DEFINITION

As the first step to setting up an HLA Federation in accordance to the HLA standards, several prerequisites need to be defined and agreed upon by all participating Federates. The first requirement is to determine the host Federate for the Federation, since the host Federate will hold the Run-Time Infrastructure (RTI) host, provide Federates a destination address to connect to, and act as the primary controller to indicate the start and end of the whole simulation execution. Although the network address of the RTI host is identified after fulfilling the first requirement, a RTI host can contain multiple Federation executions. Allocating the Federation execution name that the Federate is going to participate in is, therefore, the next pre-requisite of the simulation environment setup.

Prior to joining a Federation, one of the rules of the HLA standards is to define a common Federation Object Model (FOM) to be used that is designed according to the Object Model Template (OMT). This would provide the common data structure template for data exchanged in the HLA simulation network. Lastly, the number of Federates that are going to participate at the start of the simulation execution, or the number of Federates that the Federation host is required to wait for before beginning the execution of the interoperable simulation, have to be stated. This is to ensure synchronization of simulation time and the alleviation of the problem of earlier information not received by a

Federate that joins the simulation execution at a later time. The issue of time synchronization will be explained in further detail in the HLA Time Management (HLA-TM) implementation section.

B. EVENT GRAPH COMPONENTS

After acquiring an understanding of the requirements prior to designing the application library, the Simkit application library has to be designed generically to allow flexibility for customization of future designs. The event graph components that make up the majority of the design keep this objective in mind. The design also fulfills the HLA rules and adheres to the Application Programming Interfaces (APIs) required for interfacing with the RTI host. There are a total of four event graph components designed to manage the different aspects of setting up the HLA simulation environment as well as the simulated entities in the simulation.

1. HLA Connection Manager

The main function of the connection manager is to execute the process of setting up an HLA Federation environment and/or joining a Federation environment as a Federate. As the beginning rule of setting up an HLA simulation environment, a Federation has to be created. This provides the participating Federates a common destination to join.

The HLA Connection Manager consists of five events: Run, Create Federation, Join Federation, End Join Federation, and Publish Subscribe. As illustrated in Figure 20, the Run event, which is triggered by the execution of the SimEngine application, initiates the beginning of the setup process. Taking into consideration the pre-condition of whether the SimEngine is the Federation host or if the Federate has already joined the Federation successfully, determines the next event state that will be scheduled. Following the various iteration of creating a Federation, joining a Federation, and ending the Federation environment setup, defines the sequence of building up the HLA network and ensuring that this connection is set up properly. The three primary events, Create Federation, Join Federation, and End Join Federation,

call the standardized HLA API interface *createFederationExecution( )*, *joinFederationExecution( ),* and declaration to be *enableTimeConstrained( )* and *enableTimeRegulation( ),* respectively. The purpose of being time constrained and time regulating is to synchronize the simulation time which will be discussed in detail in the following chapters.

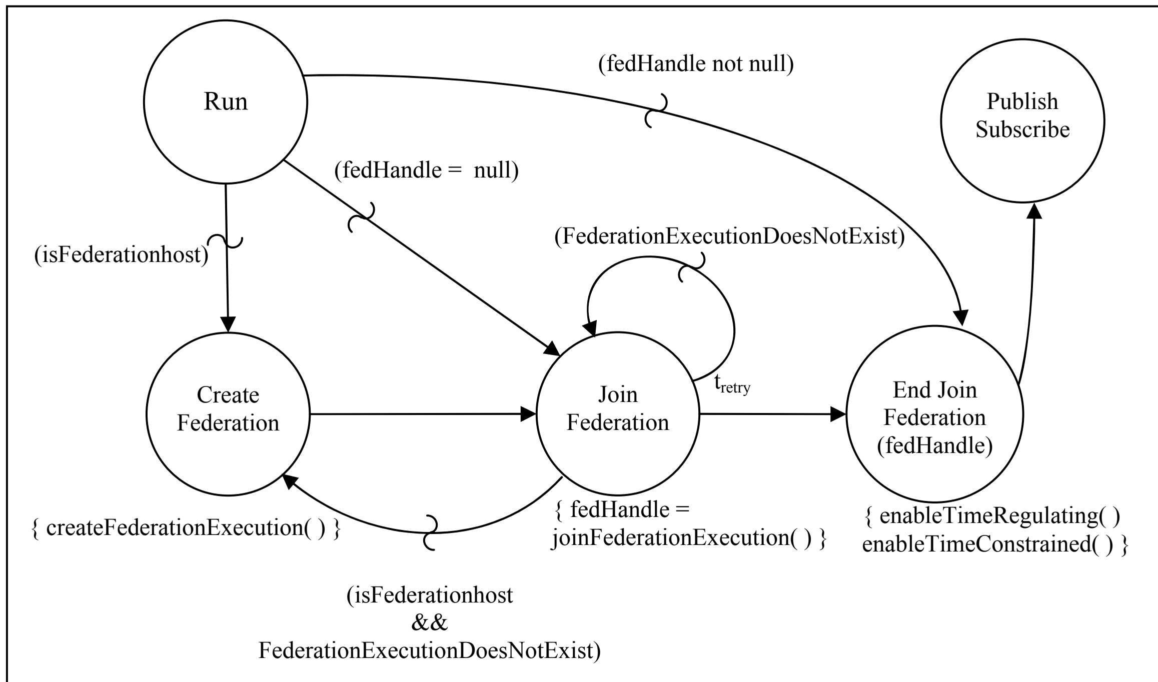


Figure 20. HLA Connection Manager event graph

Following the process of Federation creation, Federate joining and enabling the condition of time regulating and constrained, the Publish Subscribe event state is scheduled to trigger the start of the HLA Data Manager Component. This is done using the SimEventListener mechanism in Simkit.

2. HLA Data Manager

The event graph components mentioned in this application library have the objective to handle a specific aspect of the HLA simulation; hence, events might not be linked and may run individually. In conjunction with the name of this event graph component, the HLA Data Manager handles all events and processes involving the

management of HLA data that are to be sent or received. The declaration management and Management of Object Model (MOM) concept are the two services that this event graph component implements. Referencing previous discussions on declaration management and MOM service, these services involve the interfaces of publishing, subscribing, and registering of objects and interactions.

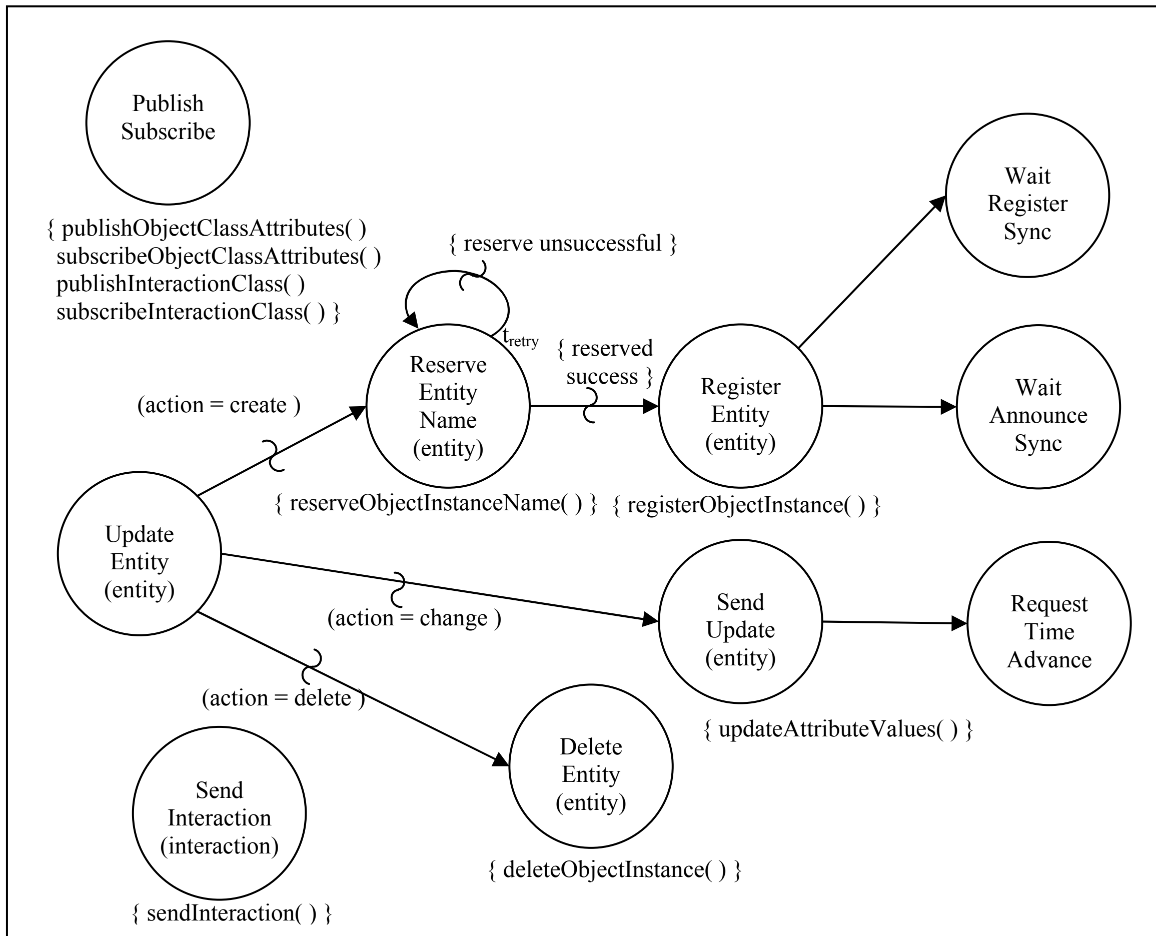


Figure 21. HLA Data Manager event graph

The HLA Data Manager was designed with three separate processes: publishing and subscription events, management of entity objects, and management of interactions. The publishing and subscription processes are part of the declaration management process. Their purpose is to inform the Federation which object classes the Federate is capable of sending and which it is interested in receiving. This reduces the processing and receiving of unwanted data. Starting from the Update Entity event, the process has

the functionality of registering an object instance which informs the Federation of its creation, sending of data updates to the Federation of registered object instances, and removing of object instances from the simulation. The last process is the Send Interaction event that is scheduled when sending a Federation event.

3. HLA Entity List Manager

This architecture design developed in this thesis study involves a different concept of entity data exchange and management compared to conventional time-stepped simulation. Entity data is sent and received in a "upon change" methodology. This resulted in the requirement of a managing component in the whole design to manage and keep track of the simulated entities, both local and HLA entities. This is the functionality of the HLA Entity List Manager event graph component. Every object and entity instance creation, changes, and deletions is recorded and stored in two mapped lists of sub-classed entity types described in the following chapter. The lists provide information of the entity objects to the SimEngine for display or processing upon request.

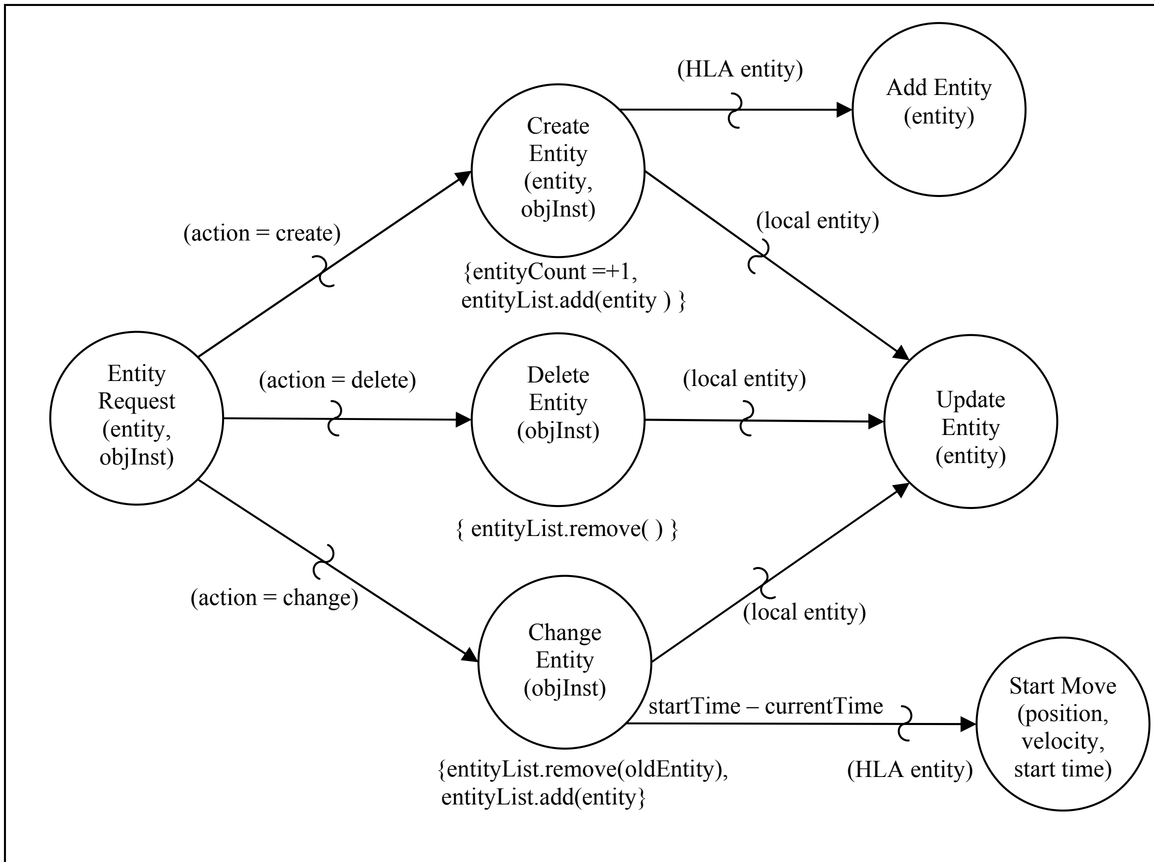


Figure 22. HLA Entity List Manager event graph

As shown in the event graph in Figure 22, the HLA Entity List Manager receives the Entity Request event and the corresponding events, according to the action requested, are scheduled. The entity list records are, subsequently, updated and depending on the nature of this entity, local or HLA, the Federation and the SimEngine is informed of this change through the Update Entity and Add Entity event state, respectively. An important event state in this event graph component is the Start Move event that marks the point-of-change of motion parameters in the simulation. The event schedules the Start Move event in the sub-classed Basic Linear Mover to provide the accurate movement characteristics of the HLA entity.

4. HLA Time Manager

The last event graph component designed is the HLA Time Manager event graph component. Listed as one of the rules of the HLA standards, certain form of time

management is required to maintain time synchronization. This is especially prominent in real time simulation when the data of simulated entities are time sensitive for display accuracy and possible conflict resolution algorithm computation at all Federates. The HLA Time Manager was created for the sole purpose of ensuring time synchronization and as part of the implementation of the time management algorithm explained in Chapter IV.

The HLA Time Manager event graph component involves two mechanisms of time management: synchronization point and time advance algorithm. The synchronization point mechanism is a series of message exchanges between the Federate and the RTI. The start of the mechanism is initiated by the Federation host, which keeps track of the number of Federates joined to the Federation. When the Federation host records sufficient number of Federates joined to the Federation and registration of synchronization point has not been initiated before, the Wait Register Sync event schedules the Register Sync Point event to begin the synchronization process. At this point of the message exchange, all Federates in the Federation are "waiting" in the Wait Announce Sync event. They are pending to be signaled by the RTI to request confirmation of their state of readiness. The Sync Achieved event state is scheduled to complete the Federate's portion of readiness confirmation. While the Federate enters a last waiting event, Wait Federation Sync, the RTI will complete this synchronization process with a Federation Synchronized message. The replies of the Federate ambassador will feedback the status of this message exchange to the HLA Time Manager.

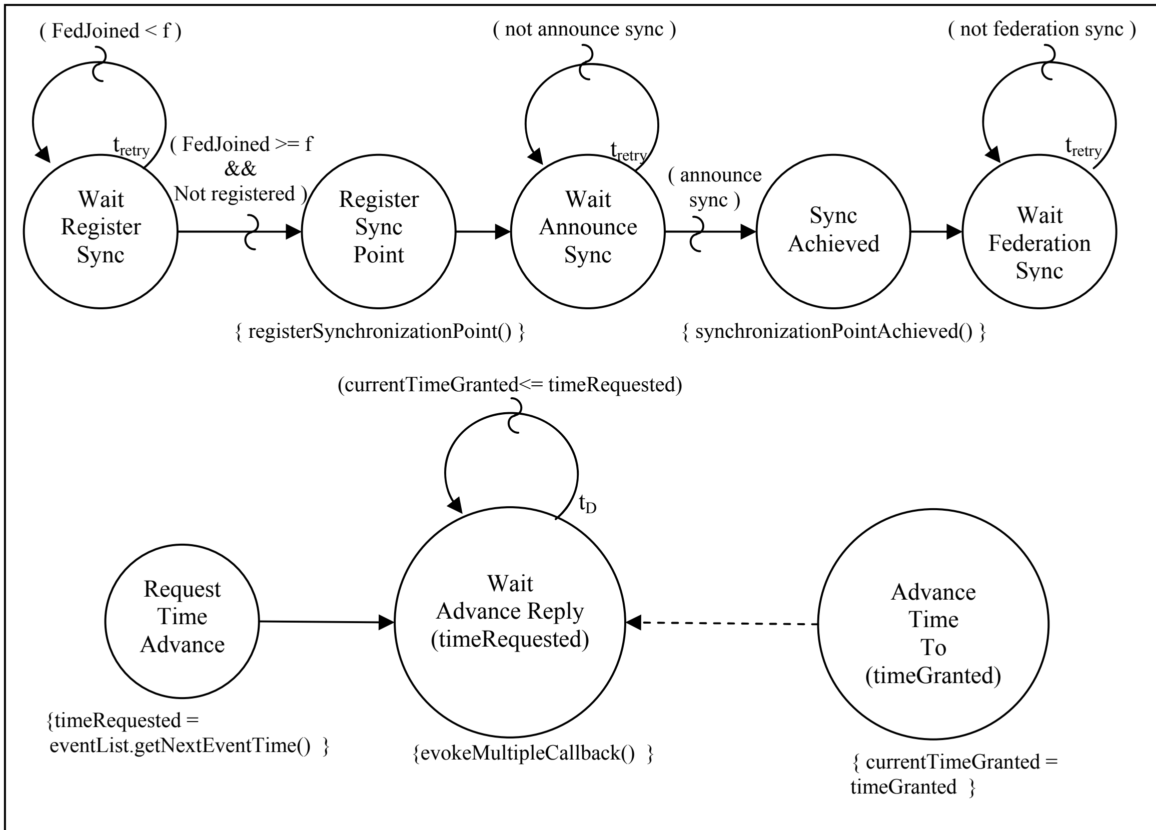


Figure 23. HLA Time Manager event graph

The other mechanism in this event graph is the process of requesting time advancement and of keeping track of the actual amount of time granted to advance. The amount of time requested to advance is obtained from querying the event list for the time of the next event that will impact the overall simulation. This means local events are not placed into consideration. Only events that will impact the simulated entities are of interest in this situation. The Wait Advance Reply event will then be scheduled in a periodic time to query the RTI for the time advance grant. This is the only event that mimics a time-stepped characteristic. Upon a time advancement grant, this periodic poll is cancelled by the Advance Time To event state.

C. SUB-CLASSED COMPONENTS

In the design of the application library, there are several extended classes instantiated to represent objects and simulated entities exchanged within the component. These components contain crucial data of the entities, motion characteristics, and the events scheduled to be executed.

The HLA Entity class is a sub-class of the Simkit SimEntity class. Every instance of this component represents a simulated entity in the Federate simulation, regardless of local or HLA. This instance of the HLA Entity class will hold the most updated information describing the simulated entity in terms of its movement characteristics, name, representative mover instance, and its action status.

The next sub-classed component is the HLA Basic Linear Mover, which is an extension of the Basic Linear Mover in the Simkit library. To compute the location of a HLA entity during every request of the SimEngine, there is a need for the application library to have the ability to do some simple internal computation. The Basic Linear Mover class in the Simkit library requires an End Move event state to be scheduled to predict when the next change of movement characteristics. This behavior is not necessary in the HLA Basic Linear Mover as it is assumed that the HLA entity will behave in the recorded movement behavior infinitely till point-of-change. In this situation, when the point-of-change will occur is not known to the Federate as the event has not happened. The HLA Basic Linear Mover was, therefore, simplified to compute the location information based on the stored data at point of request.

The last sub-classed component is the event list in Simkit. To differentiate and expose more functionality of the event list in Simkit, the HLA Event List was created. An event list holds and maintains all the local and generic critical events of the Federate simulation. It is important to avoid large extensive modification to the event list during simulation as this would cause unexpected error. The HLA Event List balances between data protection and exposure. The purpose is to minimize the required information, such as time of the next critical event and time query.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. INTERFACING APPLICATIONS

Up to this point of designing the resulting application library, it consists only of description and explanation of the architecture for the processes and functions within the application library. There are missing interfaces to link these processes to exterior components, such as the Run-Time Infrastructure (RTI) host and the SimEngine. This chapter explains the design and implementation methodology of these important interfaces supporting the operations of the application library. Without these interfaces, any application developed using the application library will not be able to communicate to the external components and fulfill some purposes of the study.

A. HLA FEDERATE AMBASSADOR

The HLA Federate ambassador is one of the required interfaces that enable Simkit to be linked to the RTI and classified as High-Level Architecture (HLA)-compliant. The RTI ambassador serves the purpose of sending messages to the RTI with the standardized Application Programming Interfaces (APIs). The complementary interface is the Federate ambassador that allows reception of messages from the RTI. Similar to all interfaces within the HLA standards, all the interfaces and APIs that call in the Federate ambassador are standardized implementations. The Federate ambassador class in the RTI interface library was, thus, implemented and sub-classed to create the HLA Federate ambassador component.

There is an extensive list of function calls in the Federate ambassador class. Not all the methods, however, are utilized in the design of the application library. The Federate ambassador class of the RTI requires a full implementation of all the function callbacks. This was overcome by implementing a null class, NullFederateAmbassador, to do nothing for all the function callbacks. The HLA Federate ambassador component, subsequently, sub-classed this null class and implements only required function callbacks to handle received messages. Appendix A lists the implemented function callbacks of the HLA Federate ambassador and their corresponding functions in the overall design.

B. HLA DATA ENCODER HELPER

A Dynamic Link-Compatible HLA API Standard for the HLA Interface Specifications is defined in the Simulation Interoperability Standards Organization (SISO) standardization process for HLA interoperability. Compliance with the APIs in this specification will permit simulation developers to interchange link compatible HLA RTIs without recompiling Federate source code or re-linking Federate object code with Dynamic Link-Compatible (DLC) RTI libraries [20]. As a part of this DLC API standard, an encoder class is required to ensure that the data types of the correct properties are encoded and decoded accurately in the HLA Federation Object Model (FOM) objects sent and received, respectively.

The design of the HLA Data Encoder Helper implemented is based on a Java Linked-Compatible (JLC) library of the RTI. The JLC APIs serves as a code wrapper around the original C++ object code. In terms of data size and coding, there is a possibility that the data type defined in the Java code is different from the C++ object code. The HLA Data Encoder Helper aims to bridge this gap and, at the same time, conform to the HLA 1516 Interface Specification standards definition of data type [20]. The helper class will directly pack the Java data type into the HLA object class structure and send out through the RTI ambassador APIs. The HLA messages received via the Federate ambassador is also unpacked using this helper class and stored into the respective object class structure listed in Appendix B.

C. HLA SIMKIT API

One of the objectives in the design of the application library is to provide a means for a SimEngine to connect to an HLA environment without the need to have any setup or implementation of the HLA standardized APIs. To achieve this functionality of the application library, it is necessary to have a simple and easy-to-implement interface. This interface class was designed and created based on the application library. The HLA Simkit API interface class, besides implementing this feature, encompasses all the internal APIs required for interfacing the Federate ambassador and the application

library. The purpose of this approach of encompassing all APIs is to provide a central location of all APIs, internal and external, for ease of reference.

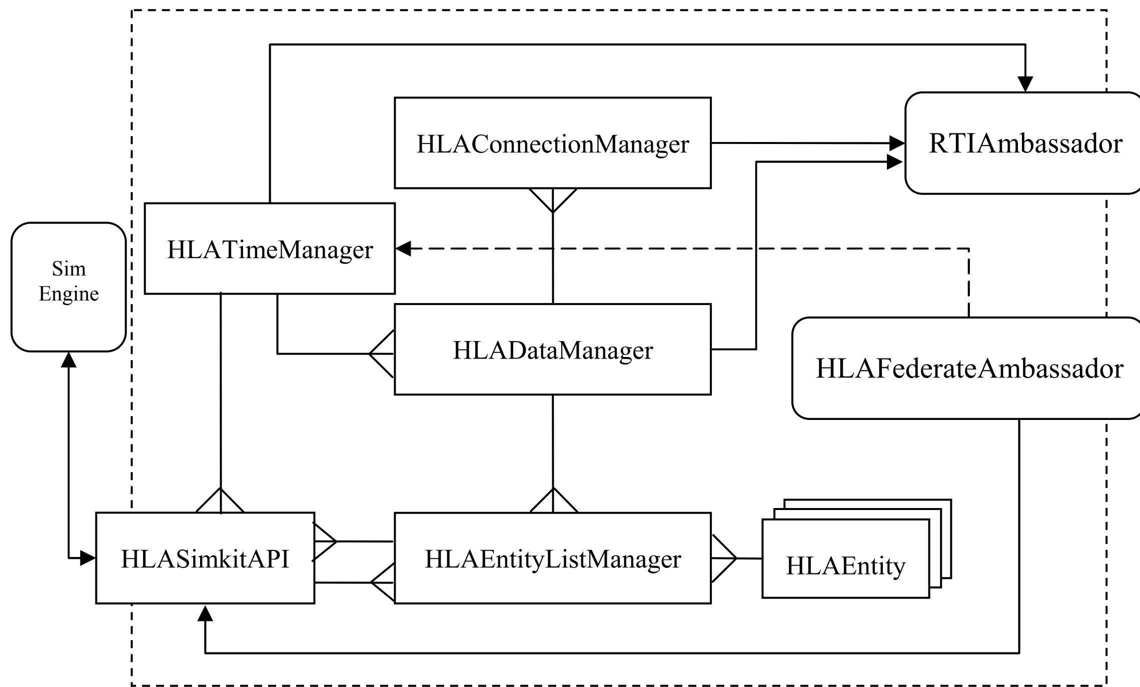


Figure 24. HLA Simkit API class event listening mapping

The HLA Simkit API class functions like an individual application that creates instances of all the components, subclasses, and interfaces of the application library when instantiated. As illustrated in Figure 24, the event listening mapping is, subsequently, created to link the components and to ensure that the proper event interactions are in place. In addition to implementing the event listening mechanism, the HLA Simkit API provides a series of interfaces between the SimEngine and the application library. These interfaces are responsible for providing the SimEngine the means to manage the local and HLA entities through simple function calls, such as create, update, and delete. Another important interface that is also included in this series of interfaces is the update request interface. When an HLA Entity data is required by the SimEngine for display or computational purposes, the SimEngine requests an update through this interface. The HLA Simkit API replies via a SimEngine pre-defined API, *doReplyPing( )*, with the latest computed location information.

The functionality of the various components was explained in Chapter V, while the internal interactions between components of the application library and external interfaces to the SimEngine and HLA host will be described in the following sections. The interfaces and event listening mechanism created in the HLA Simkit API class will be illustrated with respect to their functionality in environment setup, entity management, and time management.

1. HLA Environment Setup

As described in Chapter V, the creation of Federation host and joining to the Federation Execution as a Federate participant is triggered by running the SimEngine application. After the HLAConnectionManager component creates the Federation and joins the Federation, the setup process is handed over to the HLADataManager to handle publishing and subscribing events. This handover is performed through the event listening mechanism by having the HLADataManager listen to the Publish Subscribe event. Although the publishing and subscribing of object models is part of the setup process, this is done in the HLADataManager component. This is because the object handles that are returned when declaring that the object structure are used in subsequent events of sending object creation and data change updates. It also provides clear categorization of the functionality of event graph component, i.e., the HLADataManager is responsible for data updates of entity objects.

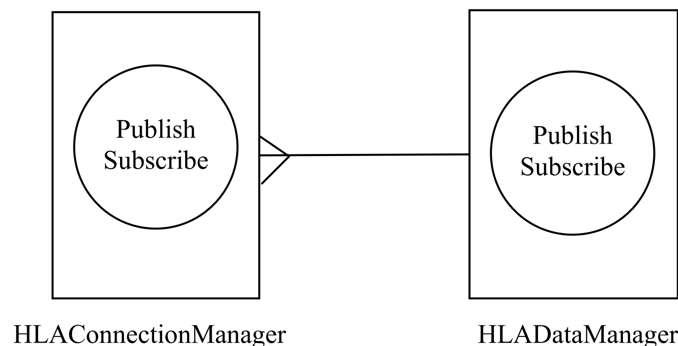


Figure 25. Environment setup interface through event listening

2. Local Entity Management

The management of the local entities follows the publishing and subscribing events. The creation and registration of the first entity is performed when an entity requests function from the HLA Simkit API interface is called. When the *CreateEntity( )* interface method is called, a record of the local entity is created and the Entity Request event state is scheduled. With the listening mechanism in place, HLAEntityListManager, which is listening for this event, will trigger the start of the creation process. This is the same flow of events used for updating and deleting local entities within the simulation. The only difference among these processes is the action performed that is recorded in the HLA Entity. This depends on which interface, *CreateEntity( )*, *UpdateEntity( )*, or *DeleteEntity( )*, is called at the HLASimkitAPI. The sequence of events ends with the HLADataManager hearing the Update Entity event from the HLAEntityListManager and handling the event based on the action recorded in the HLA Entity object to send register, update, or delete messages to the RTI.

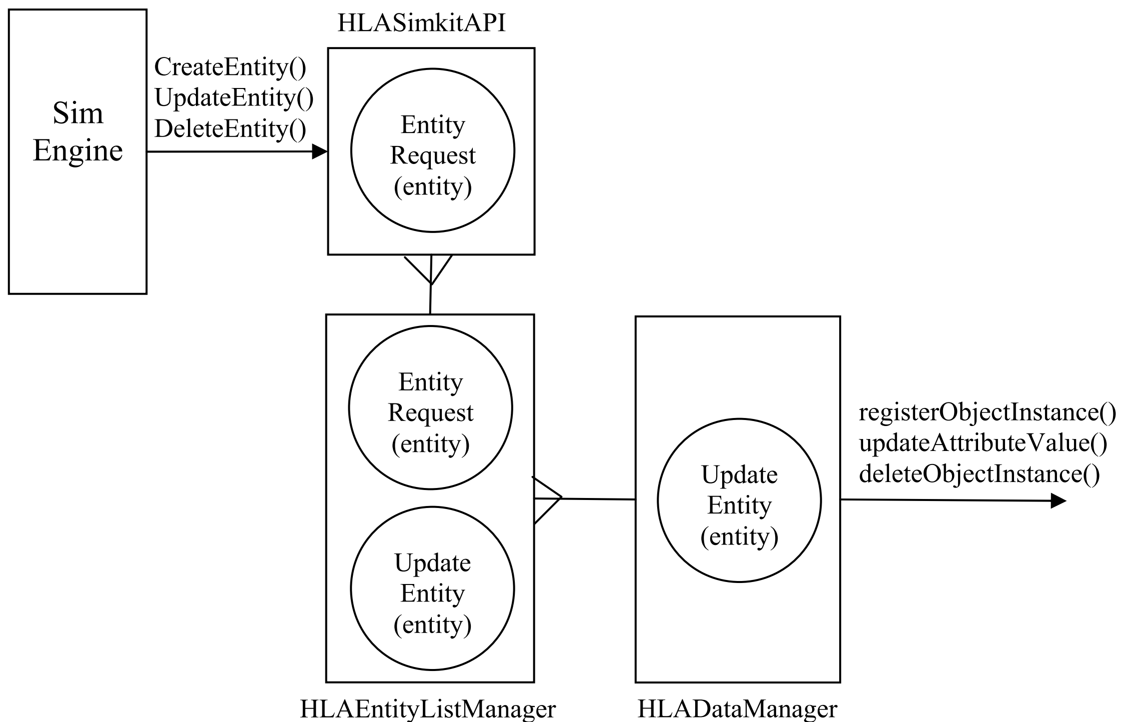


Figure 26. Local entity management interfaces through event listening

3. HLA Entity Management

In a similar manner, the entity data received from the Federation is managed and recorded into an entity list. The flow of the entity data goes through several components via the event listening mechanism. When entity data is received through the function callbacks in the Federate Ambassador, an instance of the HLAEntity type object is created and actions are set based on the callback function initiated. A *discoverObjectInstance( )* sets the create action, *reflectAttributeValue( )* is an update action and *removeObjectInstance( )* removes the record of the HLA entity.

The same interfacing function calls, *CreateEntity( )*, *UpdateEntity( )*, or *DeleteEntity( )*, are evoked to trigger the Entity Request event state. The flow of events at the end of the update process, however, diverts backwards to the HLA Simkit API instead of proceeding to the HLADataManager. This is unlike the local entity data and is because the entity data is from the Federation, so there is no need to update the Federation. At the HLAEntityListManager event graph component, the Entity Request event schedules two other event states: Add Entity and Start Move event states. The Add Entity event state informs the SimEngine of the new HLA entity through the HLA Simkit API, where a pre-defined callback function, *AddEntity( )*, is created by the SimEngine. This callback function is one of the required functions that the SimEngine application has to provide when implementing the resulting application library. The Start Move event state marks the point where the HLA entity changes its properties of movement. This event state is heard by the HLAEntity object type, which contains an HLA Basic Linear Mover element, and schedules the same event state to update the movement properties of this entity into the entity list. Updating the movement properties in the entity list provides an updated computation of the entity location when a request for update is initiated by the SimEngine.

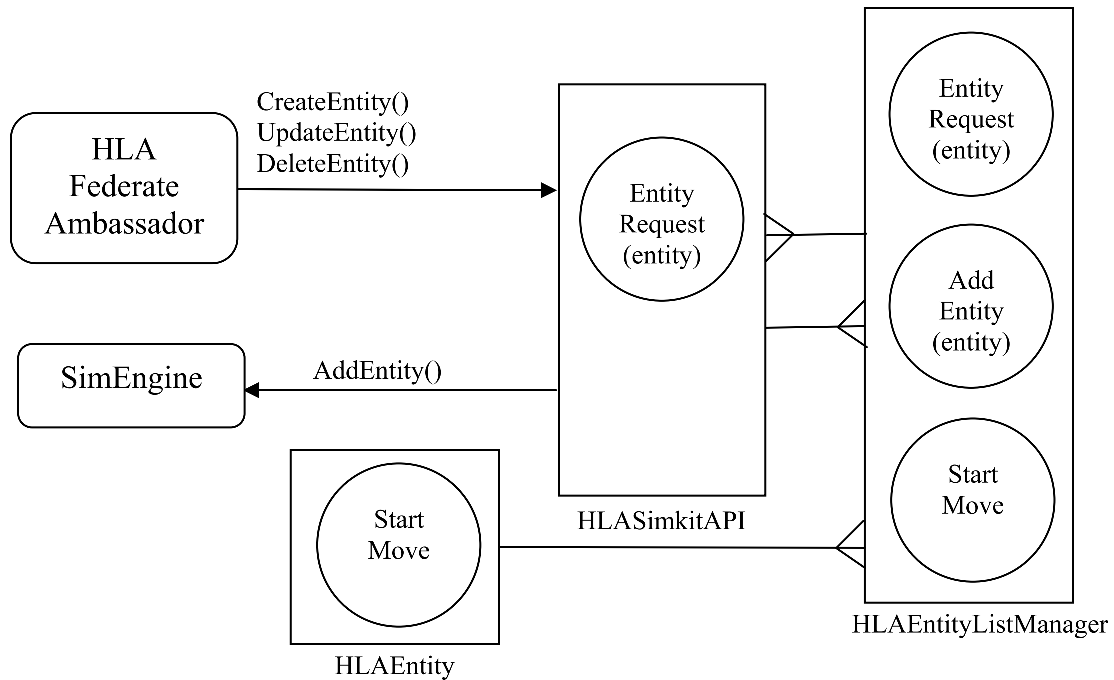


Figure 27. HLA entity management interfaces through event listening

4. Time Management

Chapter IV had an extensive discussion about the time management properties implemented in the design of the resulting application library. The HLA Simkit API supports the HLA Time Management (HLA-TM) implementation through relaying of time advance grant messages to HLA Time Manager component and activating the time offset mechanism. This is in addition to event listening mechanism mapping. After sending a data update of the local entities, the HLADataManager event graph component schedules a request to advance in time to the next time of event occurrence. This is relayed through to the HLATimeManager, which calls the *nextMessageRequest( )* API in the RTI ambassador to submit the request. The *timeAdvanceGrant( )* reply is issued by the RTI when it has ensured that there will be no chance of Time Stamp Order (TSO) messages with time stamp less than the requested. The HLA Federate ambassador calls the *TimeAdvance( )* in the HLA Simkit API, which schedules the AdvanceTimeTo event state, subsequently, upon receiving the grant. The HLATimeManager component that is

setup to listen for this event state, executes the time advancing process as a resulting action of this event state. This keeps the record of the current Federate time within the HLATimeManager for subsequent computation of the time offset mechanism and Lower Bound Time Stamp (LBTS) check.

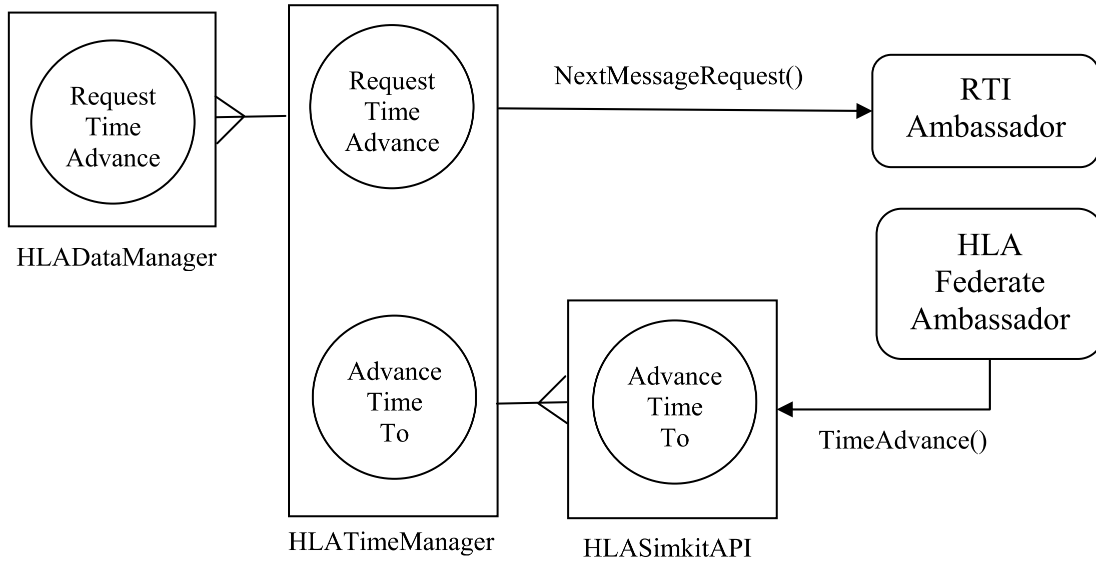


Figure 28. Time management interface through HLA event listening

VII. TEST AND EVALUATION

The objectives of this study are to reduce excessive data exchange, improve network performance, and ease the implementation requirements of a simulation into a synchronized High-Level Architecture (HLA) networked environment simulation. The resulting application library components and interfaces of this study were, therefore, put into a simple test to demonstrate whether these objectives were achieved. The test consists of implementing the resulting application library in a simple combat modeling situation with two nonHLA-compliant simulations. A numerical measurement of data traffic using a network monitoring tool, Wireshark [24], was also carried out to evaluate the network utilization during the simulation.

A. SIMULATION ENVIRONMENT

The architecture involving two nonHLA-compliant simulations was put to a test that implements the resulting application library. Although the application library is designed to be generic and compatible for most HLA design implementation, there are some specifications that are required to be standardized before any development can be done. These requirements include the Federation Object Model (FOM) to be used, the HLA standards version that the design refers, and the Simulation Engine (SimEngine) that will be used on the user interfacing ends of the virtual environment simulation.

1. Real-Time Platform Reference FOM

The FOM defines the objects and interactions that will be exchanged in the Federation. It enables a common understanding of the fields in a message packet that are exchanged within the Federation. This standardized template is one of the main criteria in a simulation to be declared as HLA-compliant. The FOM, as stated in the HLA rule, is supposed to be designed in accordance to the Object Model Template (OMT). Many vendors and sources of HLA simulation developers have customized their own proprietary versions of the FOM to be used in the simulation they have designed. This drives the need to create a common version of the FOM that would encompass most of

the parameters involved in military applications simulation in an object oriented hierarchical manner. The Real-time Platform Reference Federation Object Model (RPR FOM pronounced "reaper FOM") was designed to organize the Protocol Data Units (PDUs) of Distributed Interaction System (DIS) into a robust HLA object classes and interactions [21]. Appendix C shows the hierarchical relationship of the full list of RPR FOM objects class structures. Among the list of object classes in the RPR FOM are parameters and fields that describe an entity from a general classification of Base Entity down to the specific platform domain type, such as aircrafts, ground vehicles, and surface vessels. There are also object classes that represent equipment on board an entity, such as Embedded System, Emitter System, and stationary environment objects like mine field and craters. These objects form the main composition of the FOM to fulfill the requirements of modeling dynamic moving objects to static terrain environment objects.

In this study, to simplify the testing process and to provide a standardized model template, the RPR FOM version 2 draft 17 was used. This version of the RPR FOM was used because it is the latest object class definition and a reasonably well-rounded FOM that describes most general cases of entity configuration. The Base Entity object class is the top level class structure that contains movement parameters, such as world location, velocity, and acceleration information, required in this study. Figure 29 shows some of the possible objects of interest, sub-classed from the Base Entity Object, used in normal combat operations that can be expanded when necessary. For a simplistic proof of concept and verification of the success of this study, only the parameters in the Base Entity object class is used.

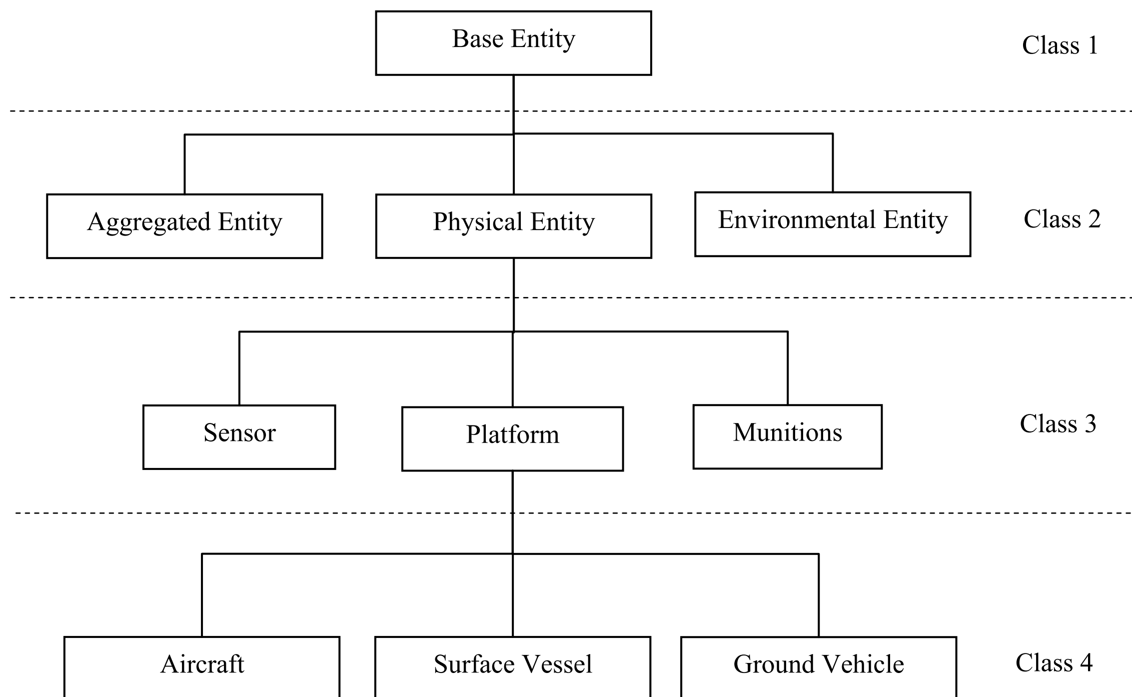


Figure 29. Base Entity object class structure

2. Run-Time Infrastructure

In the evolution of the HLA, several versions of HLA standards were developed. In accordance to the HLA standards definition, the Run-Time Infrastructure (RTI) coherently has to improve its design to be compatible to the IEEE definitions of the HLA standards. An RTI 1.1 release was originally planned to correspond to version 1.2 of the interface specifications, but the high frequency of specification and software releases and discrepancies in the release numbering schemes proved to be complicated [13].The RTI 1.3 was set as the initial definition and implementation of the RTI. RTI 1.3, however, needs to improve as technology advances and HLA standards specified by the Simulation Interoperability Standards Organization (SISO) change. In chronological order of release, this need to change resulted in the successors of the RTI 1.3: RTI 1.3-NG [13], RTI 1516, and RTI for HLA Evolved.

The RTI 1516 standard was chosen in this study to be the underlying standard for RTI implementation. This choice was made because RTI 1516 is considered as the most recently matured standard that is used constantly by the standards community in the HLA simulation arena. Although HLA Evolved is the latest standard released by SISO, it is still quite far from having widespread RTI implementations. Another reason for using the RTI 1516 is due to the Dynamic Link Compatibility (DLC) capability stated in the IEEE 1516 standards [20] that enabled the development in the Java Linked-Compatibility (JLC) libraries. The JLC definitions provided the interfaces between the RTI and Simkit, which is the simulation tool written in Java, driving the architecture design of this study.

3. Simple Movement Detection Simulation

The choice of nonHLA simulation application at the two ends of the simulation architecture, as mentioned in Figure 15, is purely based on the conditions of availability and ease of creating a simulation. The length of time available to be committed to this study is limited. It is, therefore, necessary to use any existing simulation tools, or ready-to-use applications, to perform a simple real time simulation. Thus, two instances of the Simple Movement Detection (SMD) model were selected to perform the simulation at the two ends of the nonHLA simulation architecture. The SandBox animation library in Simkit was used, in conjunction with the SMD, as the means of animating and displaying the movement of the entities. These two libraries are readily available in the Simkit application library and have been proven to work with the Discrete Event Simulation (DES) concepts implemented.

In addition to their availability, the SMD and SandBox library classes are built based on Simkit. This provides compatibility between the HLA Simkit API class and the SMD. For instance, it is not necessary to set up the update request loop between the SimEngine and the HLASimkitAPI classes since all events are scheduled to the same event list and the update request is performed by the ping thread. This simplifies the development process of creating the calling functions and callback methods required at the simulation side to interface to the HLASimkitAPI.

B. SCENARIO

This study is targeted at enhancing the ability to conduct a networked simulation under a HLA environment and at embracing the DES concepts to achieve network efficiency. It also has the purpose of providing an easy and efficient methodology of conducting simulation training and analysis over the HLA network, in particular military combat operation scenarios. To effectively demonstrate that the usage of the resulting application library meets these objectives, a combat scenario, involving a bomber and a patrolling aircraft, was used.

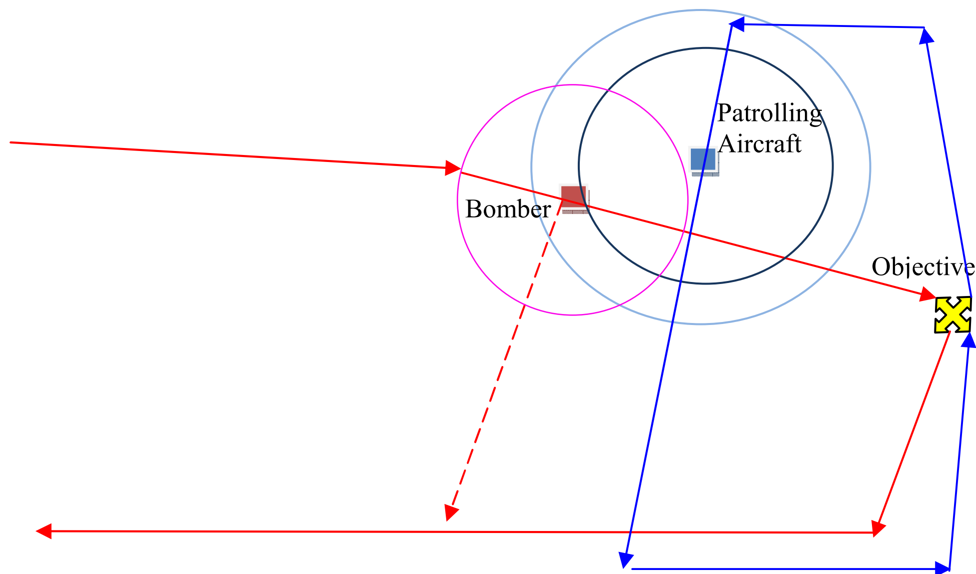


Figure 30. Simple combat scenario involving a Bomber and Patrolling Aircraft

The scenario illustrated in Figure 30 depicts a simple attack and defend combat scenario. From the attacker's perspective, a fighter bomber, represented by the red icon, conducts a bombing mission towards the objective in a flight path indicated by the solid red arrows. In the defender side, an aircraft, the blue icon, equipped with detecting sensors, is patrolling near the objective as the protecting force. The two circles, in light blue and black, are the maximum range and effective detecting circumference of the sensor onboard the patrolling aircraft.

The scenario begins with the bomber flying towards the object in the indicated flight path at a velocity of 150 knots (nautical miles per hour), which is approximately 230.4 miles per hour (mph). The patrolling aircraft is deployed from the objective to conduct a Combat Air Patrol (CAP) according to the patrol flight path, indicated by the blue arrows. The bomber proceeds according to the pre-planned flight path and enters the range of the sensor of the patrolling aircraft. The patrolling aircraft tracks and locks onto the attacking bomber to perform a defensive action of possible engagement. Given that the bomber is equipped with some form of Electronic Warfare (EW) system that notifies if defensive action is taken by the aircraft, it aborts its mission and performs an evasive maneuver.[5] The bomber flight path changes and flies in a returning flight path. This is indicated by the dotted red arrow. The patrolling aircraft breaks from its defensive reaction and continues on its CAP until it returns to the objective.

This is a situation commonly occurring in combat operations and is deemed sufficient in this study to provide several proofs of the concepts adopted. The concepts of HLA compliancy in Simkit, updating of entity data "upon change," internal computation of entity location using the equation of movement, and assessing the overall network performance can be evaluated using this scenario.

C. IMPLEMENTATION

This section of the chapter explains the implementation of the resulting application library. After defining the environment parameters and designing the scenario to simulate, there is, as mentioned in Chapter VI, a need to design the interface requirements of the SimEngine. An initial analysis of the messages exchanged when simulating the scenario is conducted.

[5] This reaction is purely for illustration purposes and does not represent normal reaction in combat operations.

1. Interface Implementation

In Chapter VI, the design of the interfaces provided to external application, the SimEngine in particular, requires some processes to call the interfacing methods for managing local entities and callback functions to be defined for managing replies and updates of HLA entities. As explained in previous sections of this chapter, the SMD model from the Simkit application library is used to perform the role of the SimEngine. There is no need to specifically call the interfacing function to manage the local entities as the events involved, local and global, are scheduled to the same event list. Instead, this allowed the event listening mechanism to be setup as the interface. The SMDHLASimkitAPI[6] Java class, an extension of the HLASimkitAPI class, which is designed to hold all the interfacing APIs, is created to setup these event listening mechanisms. These event listeners are created to specify the corresponding HLASimkitAPI interface to execute with respect to specific event states of the SMD simulation.

[6] There are multiple approaches to implement the application library. These differ according to designer implementation. Sub-classing the HLASimkitAPI is just one of these ways.

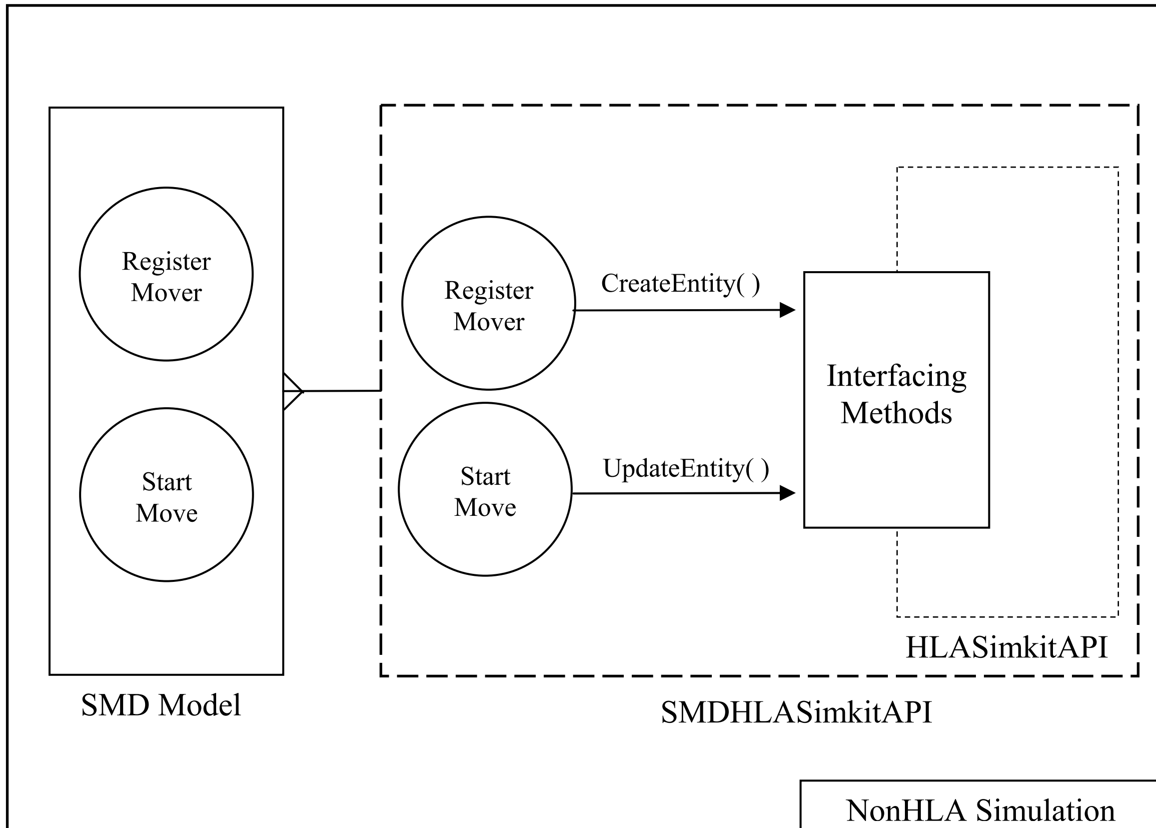


Figure 31. NonHLA Simulation implementation with SMD model

Figure 31 shows a typical implementation of the resulting application library making use of the Simkit library to perform the simulation, event management, and event listening mechanisms. The SMD model consists of several event graph components that allowed it to achieve simulation of the entities' movements. They are the mover manager that handles waypoint movements of the entity, referee and mediators that determine the detection events between entities and sensor, and, lastly, the SandBox classes that enable animated displays of the simulation with Java graphical components [5]. From the implementation of the SMD simulation model, there are two event states of particular importance when interfacing with the HLASimkitAPI. They are the Register Mover and the Start Move event states.

The Register Mover event state is scheduled when an entity mover component is created in the SMD simulation model. It declares to all interested event graph components, such as the referee and mediator, of the existence of the entity. This is to

allow the referee, the mediator, and the Sandbox, to keep track of detection events scheduling and polling of entity data for graphical display. The Register Mover event state, therefore, is required to trigger the *CreateEntity( )* interface to inform the Federation of this creation. The Start Move event state indicates the start in movement of an entity. It is, hence, an important event state to be listening for, so that the flagging of the change in entity motion can be recorded and updated to the Federation.

2. Message Exchange Walkthrough

After implementing the resulting application library and the related interfaces, it is useful to conduct a "pen and paper" simulation, without the exact reference to time, to step through the simulation processes. Listing the expected messages that would be exchanged over the network and events that would be scheduled into the event list provides a clearer picture of the expected results and ease the troubleshooting of errors that might be made during the development process. The simulation can be divided into three sections: the environment setup, the time synchronization, and the actual simulation process.

After developing the SMD simulation model as the SimEngine and integrating it with the resulting application library, connecting and setting up the HLA environment for connection is the first step. Upon instantiating the SMDHLASimkitAPI class, the connection setup begins. The sequence of creating an HLA environment starts with the simulation host creating the Federation. This is followed by the local Federate joining to the Federation. While it waits for the rest of the participating Federate to join to the Federation, it executes the object class declaration. Figure 32 shows the Unified Modeling Language (UML) method of illustrating the function calls and message exchanges between the participants of the simulation. The Federation host, SMD1, and the participating Federate, SMD2, transit into a wait state for the next section of the simulation. These functions are all carried out at simulation time zero before the simulation starts.

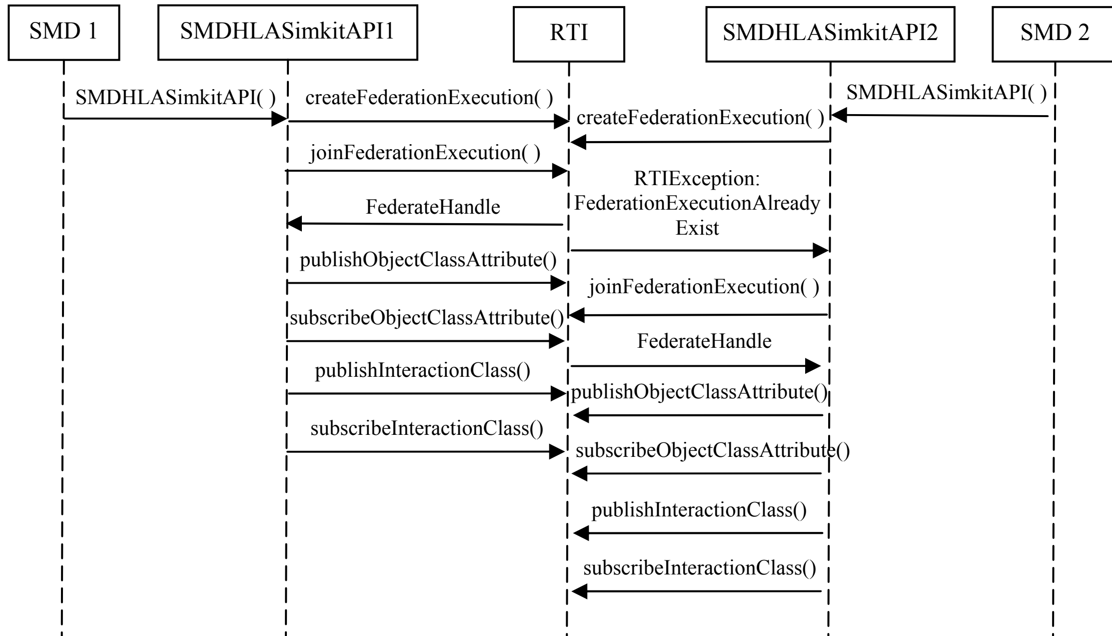


Figure 32. UML diagram of function calls for HLA connection setup

The simulation continues, subsequently, with the two SMD simulation models creating the first local entity, the bomber and the patrol aircraft, respectively. The Register Mover event state is scheduled at the corresponding event list. A *registerObjectInstance()* function is delivered to the RTI and a *discoverObjectInstance( )* delivered from the RTI to opposite Federate. This announces the creation of the entity object and acts as an indication to the Federation host that the Federate has joined successfully. In this simulation, the Federation host recorded that SMD2 has joined successfully and, thus, will begin the time synchronization process. Since the synchronization events are all scheduled with highest priority in the event list, the simulations at both sides are not able to start until the Federation is declared synchronized.

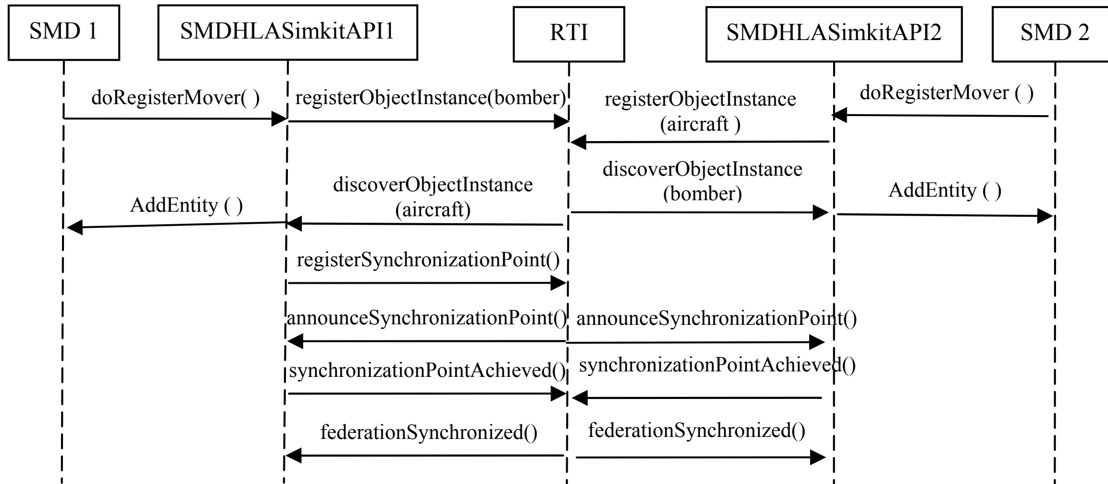
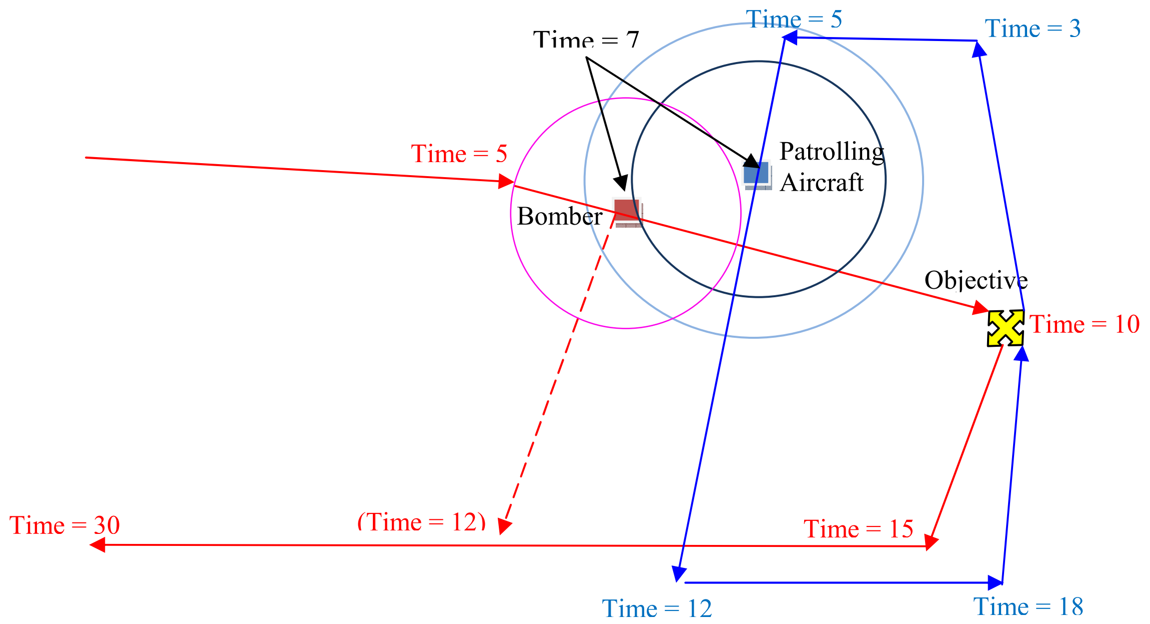


Figure 33. UML diagram of Federation time synchronization process

Declaration of the Federation as synchronized marks the completion of the setup process and the start of the actual simulation. With reference to the simulation scenario, the two entities are only created by the corresponding simulation. Their initial movement properties are not updated to the Federation execution. At simulation time equals 0, both simulations send *updateAttributeValue( )* messages to inform other Federates in the Federation of the initial data of their local entities. Subsequently, the two instances of the resulting application library implementation, SMDHLASimkitAPI1 and SMDHLASimkitAPI2, request to advance their simulation time using *nextMessageRequest( )* with time stamp 5 and 3, respectively. This updating pattern continues till the simulation time equals 5. The respective SMD simulations are doing their own time step simulations of requesting entity location updates through the Ping thread at different update rates and displaying entities on the Sandbox screen.



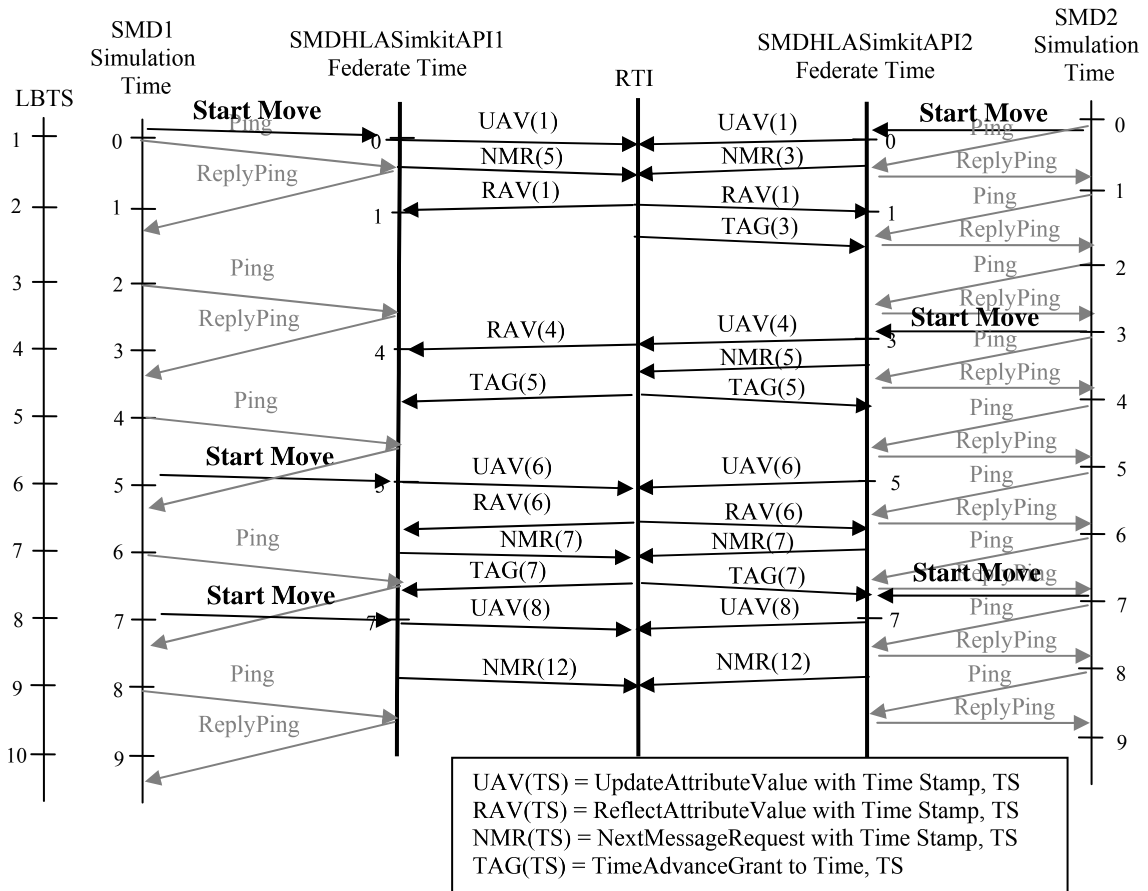


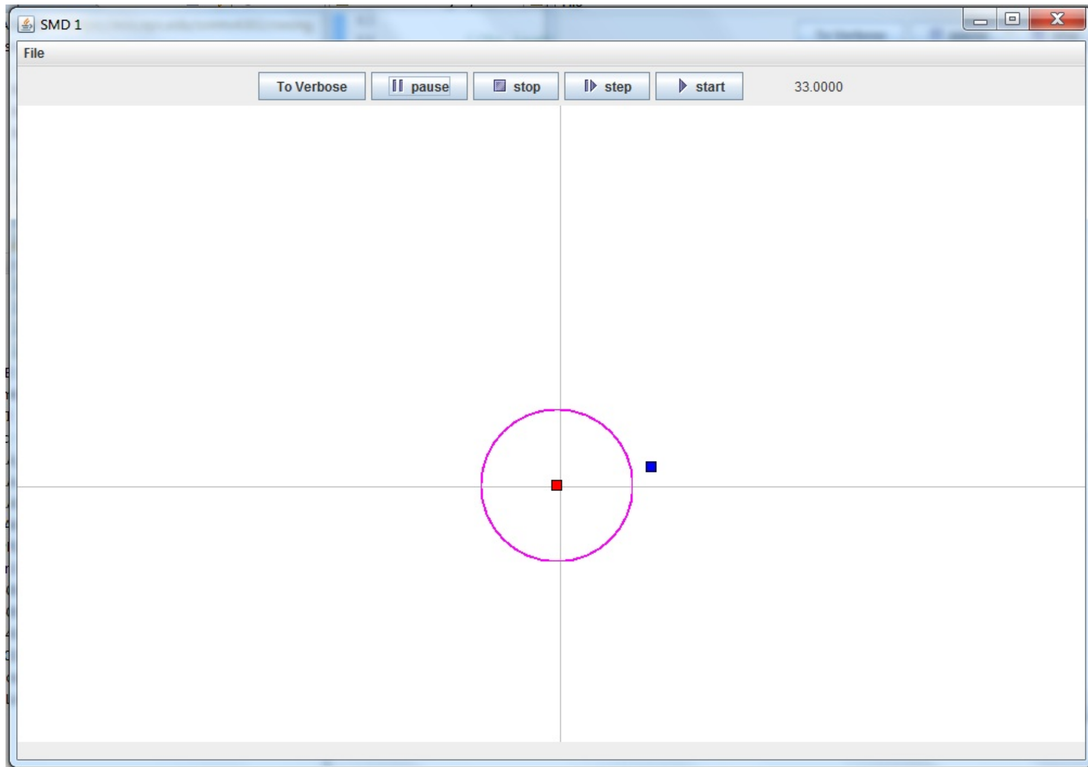
Figure 34. Simulation walkthrough of messages exchanged

As illustrated in Figure 34, at time 5, the bomber and the patrolling aircraft have the next point of change at 10 and 12, respectively. After receiving the updates occurring at time 5, the detection model of the SMD simulation schedules a Detection event state into the event list. This triggers the reaction of the bomber to abort its mission due to the patrolling aircraft defensive action. At the same instance of time, the next points of change for both simulations are removed from their individual event lists. A point of change at time 7 is scheduled into their event lists for the bomber's evasive action to return and the patrolling aircraft to prepare for engagement. Eventually, the patrolling aircraft continues its patrol route as the bomber has retreated, resulting in the *nextMessageRequest( )* with time stamp 12 for both entities. The simulation completes with the patrolling aircraft returning to the objective in the last update of movement properties at time 18.

D. RESULTS

Through simulating the mentioned scenario under the indicated HLA environment, the results of the simulation were gathered and analyzed. Through the results obtained, the data exchange reduction objective can be verified. Using two separate computer platforms, each executing an instance of the developed simulation at different update rates, the HLA-networked simulation between two nonHLA-compliant simulations was conducted. In this test, the MAK RTI [23] was used as the RTI host.

Figure 35 shows the resulting graphical display at each of the SMD simulations at the same time of 33.00. From the displays of the separated SMD simulations, the HLA data exchanges are shown to be successful and synchronized. This satisfies the objective of enabling ease of implementing nonHLA simulation into a HLA environment with the use of the resulting application library. It also shows that the methodology of using the HLA Time Management (HLA-TM) to perform a synchronized event-driven real-time simulation is possible and achievable. The difference in update rate and computer platform performance did not affect the synchronization of the simulation. It proves that the resulting application library achieved removing the possibility of problems occurring when linking simulations of different performances.



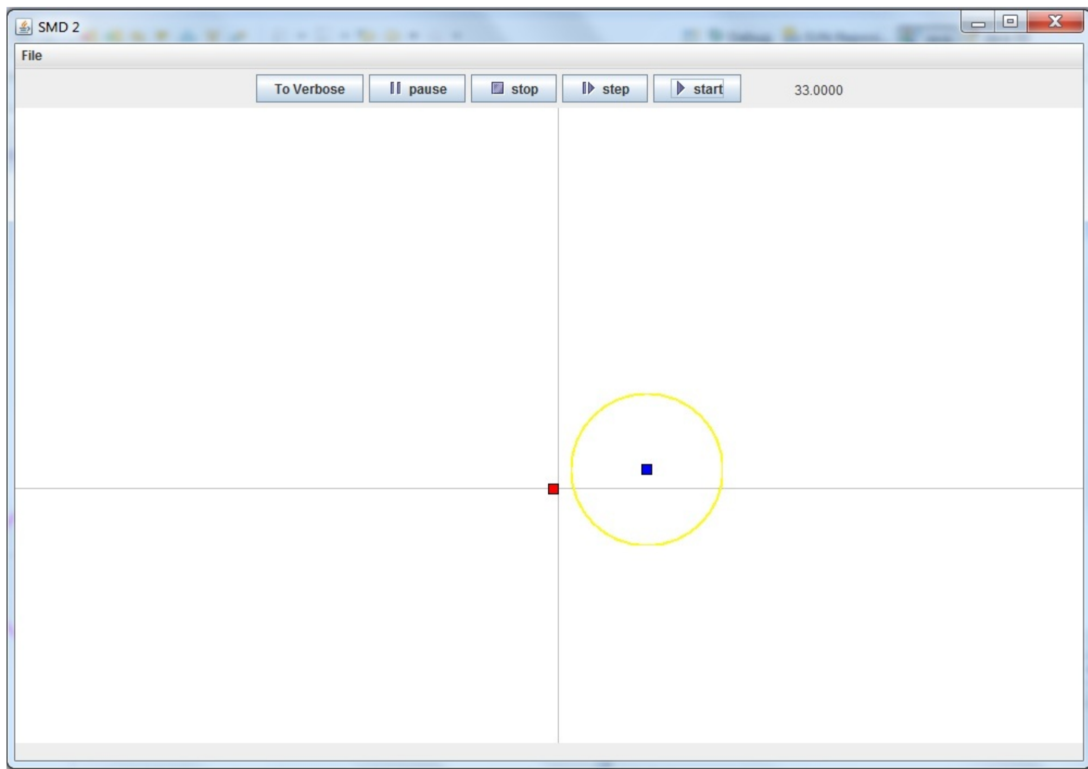
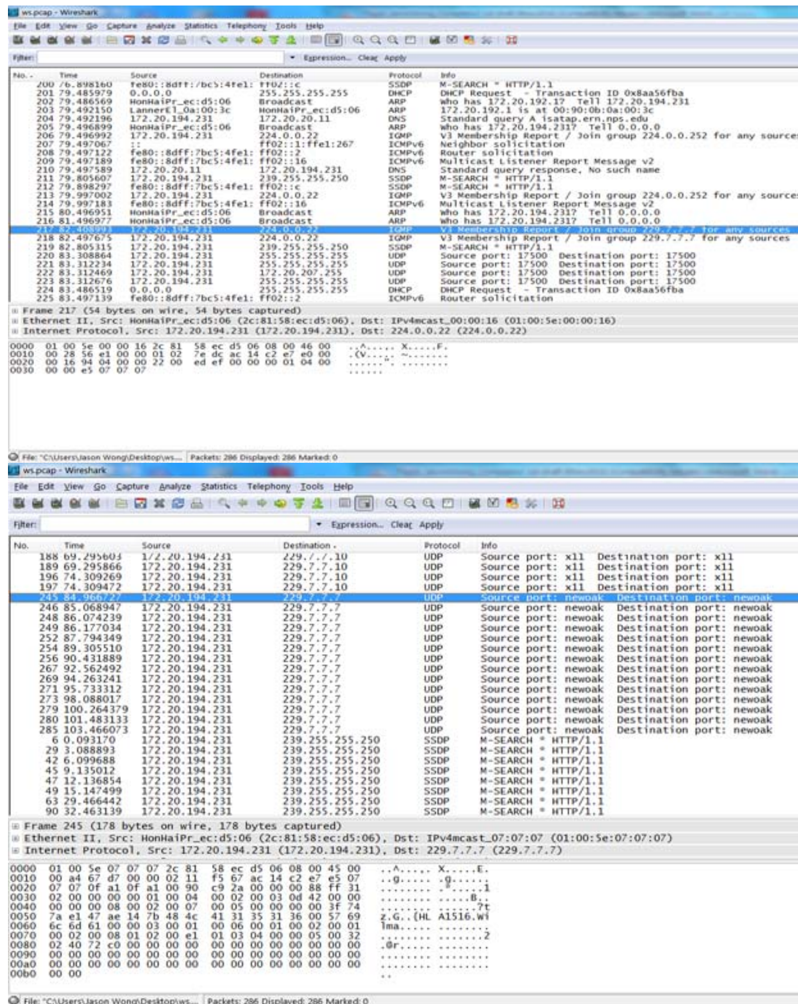


Figure 35. SMD simulation display of the synchronized simulation

During the execution of the simulation, the network monitoring tool, Wireshark [24], was activated in the Federation host computer SMD1 to track and record the messages exchanged over the network. A comparison between the number of the messages captured by Wireshark tool when simulating with the resulting application library implementation and a simple calculation of the expected count in a conventional time-stepped HLA simulation, shows that there is a significant reduction of messages exchanged.



	Time-Stepped Simulation	**HLA Simkit Application Library**
IGMP messages for multicast set up	2	2
UDP messages for data exchange	30 updates per sec x 18.499346secs ≈ 555	14

Figure 36. Wireshark records of messages and comparison of results

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. CONCLUSION

Traditional time-stepped simulation in a High-Level Architecture (HLA) networked environment dominates most of the current HLA-compliant real-time simulation processes. The problems of excessive data exchange, increasing network latency, and incompatibility of simulation of different performance to conduct synchronized simulation, were noted during the initial research of the study. This thesis study was aimed at improving, if not resolving, these problems. Before designing a possible solution, discrete Event Simulation (DES) paradigm and dead reckoning methodology was researched to provide more insight. The result of the research and development was the resulting application library and implementation architecture.

The proposed architecture and methodology of performing an HLA event-driven simulation, with the assistance of the concepts adopted from DES, was proven useful. Applying the HLA Simkit Application Programming Interface (API) library and the various event graph components enabled Simkit to be HLA-compliant. The DES concept of scheduling events into the future and the Simple Movement Detection (SMD) model of computing movement information in advance led to the achievement of the objectives identified at the beginning of the study. The study of implementing Discrete Event Simulation (DES) concepts in a High-Level Architecture (HLA) environment concluded with the successful implementation of the resulting application library. Conducting the test simulation between two nonHLA-compliant simulations and, eventually, enabling them to harness the advantages characteristically designed in the resulting application library, managed to significantly improve the issues of a traditionally designed HLA-networked simulation.

Although the results of this study have demonstrated improvements to an HLA-networked simulation through the use of DES concepts, there are still unexplored improvements in the design and further verification of the capability of the application library. The current design of the resulting application utilizes only a small part of the Federation Object Model (FOM) to exchange basic information of movement and

location. Expansion of the design to utilize fully the set of object templates could prove to be beneficial to perform an all-rounded simulation. A suite of interactions in the FOM would be helpful to achieve proper action and reaction algorithm to the simulations. An unexplored arena of this domain is experimenting with the DES methodology to execute close distance and intensive maneuver combat situation, such as the dog-fight scenario in air combat operations. In the current state of the design, these situations of high frequency changes of motion deemed to be inconclusive and possibly detrimental to the aim of reducing excessive data exchange.

Another recommendation for subsequent work is to test the boundary of this methodology of using DES in an HLA-networked environment. Developing a more data intensive simulation and integrating it with the resulting application library is another recommendation. This would create a load-test situation, thus, evaluating its limit to remain synchronized and collecting a more substantial amount of results in network performance improvements. Lastly, the only weakness of the application library is the importance of each message exchanged over the HLA network. Every message indicates a point-of-change to object information. Since DES concepts of dead reckoning are used to compute the location of the object, each message becomes crucial to the simulation. Losing a message during the exchange of data would cause the simulation to fail. Although reliable transport protocol is used in the design, with increasing network load and simulation lapse, in general there is no guarantee of message transportation. Investigation into some form of message re-transmission algorithm implementation would provide an insurance of message delivery.

In conclusion, the application library developed in this study has proven to be effective and achieved the intended objectives. Although there are some weaknesses in this methodology with regards to intensive real-time simulation, it is beneficial to implement it in combination with normal time-stepped simulation. Using the resulting application library to handle pre-planned entities with simple movement behavior, such as Computer Generated Forces (CGF), in real-time simulation would significantly lighten processing load of the simulation application and simulation engine.

APPENDIX A. HLA FEDERATE AMBASSADOR CALLBACKS

Function Callbacks	Description	Possible Exceptions
public void timeConstrainedEnabled (LogicalTime arg0)	A reply from the RTI indicating that declaring to be time constrained is successful	InvalidLogicalTime NoRequestToEnableTimeConstrainedWasPending FederateInternalError
public void timeRegulationEnabled (LogicalTime arg0)	A reply from the RTI indicating that declaring to be time regulating is successful	InvalidLogicalTime NoRequestToEnableTimeRegulationWasPending FederateInternalError
public void synchronizationPointRegistrationSucceeded (String arg0)	Indicates the success of registering a synchronization point	FederateInternalError
public void synchronizationPointRegistrationFailed (String arg0,SynchronizationPointFailureReason arg1)	Indicates the failure of registering a synchronization point	FederateInternalError
public void announceSynchronizationPoint (String arg0, **byte**[] arg1)	A message from the RTI during the synchronixation process that is sent to all joined Federates to trigger synchronized declaration	FederateInternalError
public void federationSynchronized (String arg0)	Final message of the synchronization process that informs Federates that the Federation is synchronized and simulation can begin	FederateInternalError
public void objectInstanceNameReservationSucceeded (String arg0)	Successful reservation of the object name before starting to register object	UnknownName FederateInternalError
public void discoverObjectInstance (ObjectInstanceHandle theObject, ObjectClassHandle theClassHandle, String objectName)	A forwarded message to all Federates interested the specified type of object informing of its creation	CouldNotDiscover ObjectClassNotRecognized FederateInternalError
public void reflectAttributeValues (ObjectInstanceHandle theObject, AttributeHandleValueMap theAttributeValues)	An updating message of the data of the specific entity	ObjectInstanceNotKnown AttributeNotRecognized AttributeNotSubscribed

Function Callbacks	Description	Possible Exceptions
byte[] userSuppliedTag (OrderType sentOrdering, TransportationType theTransport, LogicalTime theTime, OrderType receivedOrdering)		FederateInternalError
public void receiveInteraction (InteractionClassHandle theInteractionClass, ParameterHandleValueMap theParameterValues, byte[] theUserSuppliedTag, OrderType sentOrder, TransportationType theType, LogicalTime theTime, OrderType receiveOrder)	An indication of an incoming event	InteractionClassNotRecognized InteractionParameterNotRecognized InteractionClassNotSubscribed InvalidLogicalTime FederateInternalError
public void removeObjectInstance (ObjectInstanceHandle arg0, **byte**[] arg1, OrderType arg2, LogicalTime arg3, OrderType arg4)	A deletion message of the specified entity	ObjectInstanceNotKnown FederateInternalError
public void timeAdvanceGrant (LogicalTime arg0)	A result of a next message request allowing the indicated time advancing allowance	InvalidLogicalTime JoinedFederateIsNotInTimeAdvancingState FederateInternalError

APPENDIX B. JAVA HLA SIMKIT FOM OBJECT CLASS

Object Class	Parameters	Data Type	Description
BaseEntity	entityType	EntityTypeStruct	Contains DIS coding of identifying the type of object
	entityIdentifier	EntityIdentifierStruct	Contains the specific identification of the object in the simulation
	spatial	SpatialStruct	Motion data of the object
EntityTypeStruct	entityKind	byte	Defines the kind of entity the object represents. E.g., 1 - Platform 2 - Munition 9 - Sensor
	domain	byte	Defines the domain of the entity as a land, air, or surface object
	countryCode	short	Defines the country of origin of the object
	category	byte	Defines the category of type of object. E.g., a tank of the land platform domain
	subcategory	byte	Further break down of the object type into specifc group of category. E.g., M1 Abrams Main Battle Tank.
	specific	byte	Indicates the specific model or version of the object type. E.g., M1A1 Abrams
	extra	byte	Indicates the an extra information of the object type
EntityIdentifierStruct	FederateIdentifier	FederateIdentifierStruct	Indicates the ownership of the object
	entityNumber	short	Entity number ID of the object in the simulation
FederateIdentifierStruct	siteID	short	Specific numeric ID of the hardware location
	applicationID	short	Specific numeric ID of the application in the hardware platform

Object Class	Parameters	Data Type	Description
SpatialStruct	deadReckoningAlgorithm	byte	Indicates which type of dead reckoning algorithm is used in the simulation. E.g., 1-Static, 2-FPW
	padding	byte	
	spatialStatic	SpatialStaticStruct	Information of location and orientation if the object is using static dead reckoning
	spatialFPW		Information of location,velocity, and orientation if the object is using FPW dead reckoning
SpatialStaticStruct	worldLocation	WorldLocationStruct	Location of the object in the world coordinates
	isFrozen	boolean	Indicates if object is paused
	orientation	OrientationStruct	Orientation of the object
SpatialFPStruct	worldLocation	WorldLocationStruct	Location of the object in the world coordinates
	isFrozen	boolean	Indicates if object is paused
	orientation	OrientationStruct	Orientation of the object
	velocityVector	VelocityVectorStruct	Velocity data of the object
WorldLocationStruct	X	double	x-component of object location coordinates
	Y	double	y-component of object location coordinates
	Z	double	z-component of object location coordinates
OrientationStruct	Psi	float	Roll component of the object orientation
	Theta	float	Pitch component of the object orientation
	Phi	float	Yaw component of the object orientation
VelocityVectorStruct	xVelocity	float	x-component of the object velocity vector
	yVelocity	float	y-component of the object velocity vector
	zVelocity	float	z-component of the object velocity vector

APPENDIX C. RPR FOM OBJECT CLASS STRUCTURE [22]

Class1	Class2	Class3	Class4
ActiveSonarBeam (PS)			
BaseEntity [25] (S)	AggregateEntity (PS)		
	EnvironmentalEntity (PS)		
	PhysicalEntity (S)	Platform (S)	Aircraft (PS)
			AmphibiousVehicle (PS)
			GroundVehicle (PS)
			MultiDomainPlatform (PS)
			Spacecraft (PS)
			SubmersibleVessel (PS)
			SurfaceVessel (PS)
		Lifeform (S)	Human (PS)
			NonHuman (PS)
		CulturalFeature (PS)	
		Expendables (PS)	
		Munition (PS)	
		Radio (PS)	
		Sensor (PS)	
		Supplies (PS)	
EmbeddedSystem (N)	Designator (PS)		
	EmitterSystem (PS)		
	IFF (N)	NatoIFF (N)	NatoIFFInterrogator (PS)
			NatoIFFTransponder (PS)
		SovietIFF (N)	SovietIFFInterrogator (PS)
			SovietIFFTransponder (PS)
		RRB [85] (PS)	
	MinefieldData (PS)		
	RadioReceiver (PS)		
	RadioTransmitter (PS)		
	UnderwaterAcousticsEmission (N)	ActiveSonar (PS)	
		AdditionalPassiveActivities (PS)	
		PropulsionNoise (PS)	
EmitterBeam (S)	RadarBeam (PS)		
	JammerBeam (PS)		
EnvironmentObject (S)	ArealObject (S)	MinefieldObject (PS)	
		OtherArealObject (PS)	
	LinearObject (S)	BreachableLinearObject (PS)	
		BreachObject (PS)	
		ExhaustSmokeObject (PS)	
		MinefieldLaneMarkerObject (PS)	
		OtherLinearObject (PS)	
	PointObject (S)	BreachablePointObject (PS)	
		BurstPointObject (PS)	
		CraterObject (PS)	
		OtherPointObject (PS)	
		RibbonBridgeObject (PS)	
		StructureObject (PS)	
EnvironmentProcess (PS			
GriddedData (PS)			
Minefield (PS)			

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] D. J. Price, S. Nahavandi, S. Walsh, and D. Creighton, "Linking discrete event simulation models using HLA," *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, Oct. 10–12, 2005, vol. 1, no., pp. 696–701.

[2] C. D. Pham and R. Bagrodia, "HLA support in a discrete event simulation language," *Proceedings of the Third Workshop on Distributed Interactive Simulation and Real-Time Applications*, Oct. 1999, pp. 93–100.

[3] Discrete event simulation. (October 6, 2010), *Wikipedia, The Free Encyclopedia*. Retrieved 22:56, Oct. 8, 2010, Retrieved from http://en.wikipedia.org/w/index.php?title=Discrete_event_simulation&oldid=389026003.

[4] A. Buss, "Event graph models and simkit," class notes for OA3302, Operations Research Department, Naval Postgraduate School, Monterey, California, Winter 2009.

[5] A. H. Buss and P. J. Sánchez, "Simple movement and detection in discrete event simulation," *Proceedings of the 2005 Winter Simulation Conference,* 2005, pp. 992–1000.

[6] R. Crosbie, J. Zenor, and S. Goberstein, "High-Level Architecture module 1, part 1 introduction to the high-level architecture," Cal State University, Chico, California, Nov. 14, 2001, http://www.ecst.csuchico.edu/~hla/LectureNotes/HLA_1516_M1_P1.doc.

[7] "IEEE standard for modeling and simulation (M&S) High-Level Architecture (HLA)-Object Model Template (OMT) Specification," IEEE Std 1516.2-2000, pp. i–130, 2001. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=915738

[8] R. M. Fujimoto and R. M. Weatherly, "HLA time management and DIS," *14th Workshop on Standards and Interoperability of Distributed Simulations*, Mar. 1996.

[9] T. C. Hyon and R. M. Weatherly, "The High-Level Architecture (HLA) Management Object Model (MOM) Extensions in RTI Version 1.0," *1997 Spring Simulation Interoperability Workshop*.

[10] R. Crosbie, J. Zenor, and S. Goberstein, "High-Level Architecture module 2, advance topics: management object model," Cal State University, Chico, Sept. 16, 1999, http://www.ecst.csuchico.edu/~hla/LectureNotes/Mom.pdf.

[11] R. M. Fujimoto and R. Weatherly, "Time management in the DoD High-Level Architecture," *Proceedings of the 10th Workshop on Parallel and Distributed Simulation,* IEEE Computer Society Press, Los Alamitos, Calif., 1996, pp. 60–67.

[12] R. M. Fujimoto, "HLA time management: design document 1.0.," Aug. 15, 1996. Retrieved from www.cc.gatech.edu/computing/pads/PAPERS/HLA-TM-1.0.pdf.

[13] Department of Defense, Defense Modeling and Simulation Office, "High-Level Architecture Run-Time Infrastructure: RTI 1.3-next generation programmer's guide version 3.2." Retrieved from http://sslab.cs.nthu.edu.tw/~fppai/HLA/RTI%201.3/RTI_NG13_Programer%20Guide.pdf.

[14] L. Schruben, "Simulation modeling with event graphs," *Communications of the ACM*, vol. 26, No. 11, pp. 957–963, Nov. 1983.

[15] A. H. Buss, "Component based simulation modeling with Simkit," *Proceedings of the 2002 Winter Simulation Conference,* 2002, pp. 243–249.

[16] R. M. Fujimoto, "Parallel and distributed simulation," *1995 Winter Simulation Conference Proceedings,* Dec. 1995, pp. 118–125.

[17] A. H. Buss, "Basic event graphs modeling," *Simulation News Europe, Technical Notes*, Issue 31, Apr. 2001.

[18] A. H. Buss and P. J. Sanchez, "Modeling very large scale systems: building complex models with LEGOs (Listener Event Graph Objects)," *Proceedings of the 34th conference on Winter simulation: Exploring new frontiers*, Dec. 8–11, 2002, San Diego, California.

[19] A. H. Buss, "Modeling with Event Graphs," *Proceedings of the 1996 Winter Simulation Conference,* 1996, pp. 153–160.

[20] SISO, "Dynamic Link Compatible HLA API Standard for the HLA Interface Specification (IEEE 1516.1 Version)," SISO-STD-004.1-2004. Retrieved from http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=30828.

[21] SISO, "Guidance, rationale, and interoperability manual for the Real-time Platform Reference Federation Object Model (RPR FOM) Version 2.0D17v3," Oct. 3, 2003. Retrieved from SISO Digital Library http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=17151

[22] SISO, "Real-time Platform Reference Federation Object Model (RPR FOM) Version 2.0D17," Sept. 10, 2003. Retrieved from SISO Digital Library http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=17186

[23] MAK Run-Time Infrastructure trial version 4.0, Retrieved Sept. 3, 2010, from www.mak.com

[24] Wireshark version 1.2.10, Retrieved Aug. 23, 2010, from http://media-2.cacetech.com/wireshark/win64/wireshark-win64-1.4.1.exe

[25] Richard M. Fujimoto, Kalyan Perumalla, Alfred Park, Hao Wu, Mostafa H. Ammar, George F. Riley, "Large-scale network simulation: how big? how fast?" *Modeling, Analysis, and Simulation of Computer Systems, International Symposium on, p. 116, 11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'03)*, 2003.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Arnold H. Buss
   Naval Postgraduate School
   Monterey, California

4. Donald McGregor
   Naval Postgraduate School
   Monterey, California

5. Professor Tat Soon Yeo
   Director
   Temasek Defence Systems Institute (TDSI)
   National University of Singapore
   Singapore

6. Ms Tan Lai Poh
   Senior Manager
   Temasek Defence Systems Institute (TDSI)
   National University of Singapore
   Singapore

7. Singapore Technologies Electronics (Training and Simulation Systems) Pte Ltd
   Singapore