

2005

A Framework To Model Complex Systems Via Distributed Simulation: A Case Study Of The Virtual Test Bed Simulation System Using the High Level Architecture

Jaebok Park
University of Central Florida

 Part of the [Engineering Commons](#)

Find similar works at: <https://stars.library.ucf.edu/etd>

University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Park, Jaebok, "A Framework To Model Complex Systems Via Distributed Simulation: A Case Study Of The Virtual Test Bed Simulation System Using the High Level Architecture" (2005). *Electronic Theses and Dissertations, 2004-2019*. 369.

<https://stars.library.ucf.edu/etd/369>

A FRAMEWORK TO MODEL COMPLEX SYSTEMS VIA DISTRIBUTED SIMULATION –
A CASE STUDY OF THE VIRTUAL TEST BED SIMULATION SYSTEM
USING THE HIGH LEVEL ARCHITECTURE

by

JAEBOK PARK

B.E. The Naval Academy, Republic of Korea, 1988

M.S. Western Illinois University, 1994

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Modeling and Simulation
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2005

Major Professor: Jose Sepulveda

Major Professor: Luis Rabelo

© 2005 Jaebok Park

ABSTRACT

As the size, complexity, and functionality of systems we need to model and simulate continue to increase, benefits such as interoperability and reusability enabled by distributed discrete-event simulation are becoming extremely important in many disciplines, not only military but also many engineering disciplines such as distributed manufacturing, supply chain management, and enterprise engineering, etc.

In this dissertation we propose a distributed simulation framework for the development of modeling and the simulation of complex systems. The framework is based on the interoperability of a simulation system enabled by distributed simulation and the gateways which enable Commercial Off-the-Shelf (COTS) simulation packages to interconnect to the distributed simulation engine.

In the case study of modeling Virtual Test Bed (VTB), the framework has been designed as a distributed simulation to facilitate the integrated execution of different simulations, (shuttle process model, Monte Carlo model, Delay and Scrub Model) each of which is addressing different mission components as well as other non-simulation applications (Weather Expert System and Virtual Range). Although these models were developed independently and at various times, the original purposes have been seamlessly integrated, and interact with each other through Runtime Infrastructure (RTI) to simulate shuttle launch related processes.

This study found that with the framework the defining properties of complex systems - interaction and emergence – are realized and that the software life cycle models (including the spiral model and prototyping) can be used as metaphors to manage the complexity of modeling and simulation of the system. The system of systems (a complex system is intrinsically a “system

of systems”) continuously evolves to accomplish its goals, during the evolution subsystems coordinate with one another and adapt with environmental factors such as policies, requirements, and objectives. In the case study we first demonstrate how the legacy models developed in COTS simulation languages/packages and non-simulation tools can be integrated to address a complicated system of systems. We then describe the techniques that can be used to display the state of remote federates in a local federate in the High Level Architecture (HLA) based distributed simulation using COTS simulation packages.

ACKNOWLEDGMENTS

It hardly seems possible to reach at this level without the support from many others, who have helped me in so many ways along the way, and that it only remains to thank the people who have helped me. I know that I will never be able to truly express my appreciation. I would like to thank my advisors, Dr. Jose A. Sepulveda and Dr. Luis Rabelo for their years of encouragement, patient, and support on my research in the Ph.D program. I would like to also thank the rest of my dissertation committee: Dr. J. Peter Kincaid, Dr. Charles Reilly, and Dr. Joohan Lee for their helpful comments and suggestions during the course of my Ph.D. study.

I would like to thank my colleagues in the Center for NASA Simulation Research Group: Mario Marine, and previous members: Fred Gruber, Oscar Martínez, Amit Wasadikar, Amith Paruchuri, Ann Dalrymple, Asisa Musa, and Usha Neupane for their work which were a part of this work (mainly the case study of “Factors Affecting the Expectation of Casualties in the Virtual Range Toxicity Model”).

Most of all, I will be grateful to my wife Jinsuk Na and my daughter Subin for their understanding and support throughout this effort. Without their constant sacrifice, I wouldn’t have come so far.

TABLE OF CONTENTS

LIST OF FIGURES	viii
LIST OF TABLES	x
LIST OF ACRONYMS/ABBREVIATIONS	xi
CHAPTER ONE: INTRODUCTION	1
Background of Study	1
Objective and Scope	11
Dissertation Outline	12
CHAPTER TWO: REVIEW OF RELATED TECHNOLOGIES	14
Introduction	14
Virtual Test Bed (VTB)	14
High Level Architecture (HLA)	21
Discrete-Event Simulation Languages and Packages	34
Visualization of Distributed Simulation Systems	44
CHAPTER THREE: THE HLA INTEROPERABILITY IN SIMULATION LANGUAGES/PACKAGES	50
Federate Requirements to Become a HLA Compliant	51
HLA Support in Modeling Languages and Packages	53
Implementation of The Basic Discrete Event Simulation Class in The SPEEDES Process Model	63
Visualizations in the VTB	72
CHAPTER FOUR: CASE STUDIES	84

Factors Affecting the Expectation of Casualties in the Virtual Range Toxicity Model	84
Implementing the High Level Architecture in the Virtual Test Bed	105
CHAPTER FIVE: A PROTOTYPE IMPLEMENTATION OF VTB SIMULATION SYSTEM	
.....	113
Introduction and Motivation	113
Adapting Legacy Models to VTB Simulation System Using the HLA.....	114
Integration of the Framework for Spaceport Simulation System	125
CHAPTER SIX: CONCLUSION.....	132
Summary	132
Limitation.....	134
Future Work.....	135
LIST OF REFERENCES	139

LIST OF FIGURES

Figure 1 Concept of Virtual Test-Bed (adapted with modification from (Rabelo, 2002b))	16
Figure 2 VTB System Architecture	19
Figure 3 Federate – Federation Interplay.(adapted from (DMSO, 1998b)).....	25
Figure 4 RTI Components At-a-Glance. (taken from (DMSO, 1998b))	26
Figure 5 RTI and Federate Code Responsibilities. (taken from (DMSO, 1998b)).....	27
Figure 6 SPEEDES Modeling Framework (taken from (Bailey et al., 2001))	37
Figure 7 S_Shuttle Simulation Object	38
Figure 8 Arena Modeling Environment.....	42
Figure 9 Architecture of AnyLogic Modeling and Simulation Environment (taken from (Borshchev et al., 2002)).....	44
Figure 10 RTI and Federate “Ambassadors” (DMSO, 1998a)	52
Figure 11 SPEEDES Interface to the HLA (adapted from (Bailey et al., 2001))	55
Figure 12 An example of "conversions.par"	56
Figure 13 Initial Adapter Architecture(taken from (Kuhl & Riddick, 2000))	59
Figure 14 An Example of Initialization File	60
Figure 15 Object and Message Format	61
Figure 16 Integrating HLA and AnyLogic (adapted from (Borshchev et al., 2002)).....	62
Figure 17 Class Diagram of Simulation Classes.....	67
Figure 18 An Example Sequence Diagram of Simulation Classes.....	71
Figure 19 Calculation of Population at Risk.....	75
Figure 20 Model-Animator-Scheduler paradigm (Lin et al., 1992).....	76

Figure 21 Model-Animation on the HLA	79
Figure 22 An Example model for a Remote Federate Animation	80
Figure 23 HLA Interaction Class for Declaration Management Service.....	82
Figure 24 HLA Interaction Class for Object Management Service.....	82
Figure 25 Virtual Range Toxicity Model Architecture (Sepulveda et al., 2004a).....	87
Figure 26 Launch sectors from the Eastern Range (adapted from (Sepulveda et al., 2004a).....	90
Figure 27 Exposure Response Function for HCI (adapted from (Sepulveda et al., 2004a).....	101
Figure 28 VR Simulation Interface.....	103
Figure 29 Bad Weather Occurrence (adapted from (Lebo et al., 2002))	109
Figure 30 Launch area intrusions (adapted from (Lebo et al., 2002))	109
Figure 31 Contributions to the delays and scrubs (adapted from (Lebo et al., 2002))	110
Figure 32 Overall Probability of a delay or scrub (adapted from (Lebo et al., 2002))	111
Figure 33 Distributed Shuttle Process Simulation using the DMS Adapter	112
Figure 34 The MonteCarlo Federate and The VR Federate architecture.....	115
Figure 35 Adapted Weather Expert System (WES) Architecture to the HLA	119
Figure 36 An implementation of the WES federate.....	120
Figure 37 Life cycle of the Orbiter (as an active entity).....	122
Figure 38 S_Orbiter Process (Active Entity) Logic Example	123
Figure 39 Space Shuttle Federate Architectue.....	124
Figure 40 A Prototype Implementation of VTB Simulation System.....	126
Figure 41 Interactions between the models during the federation execution	127
Figure 42 Visualization of Mission Control Room	129

LIST OF TABLES

Table 1	The DMS Adapter Interface Methods	58
Table 2	Commonly used Shuttle Propellants (adapted from (Sepulveda et al., 2004a))	85
Table 3	Representative Events in the Shuttle's Trajectory during the first 120 Seconds (adapted from (Sepulveda et al., 2004a)).....	93
Table 4	Factors affecting Ec (Sepulveda et al., 2004a)	104
Table 5	Factors affecting delays and scrubs (adapted from (Lebo & Woltman, 2002))	108

LIST OF ACRONYMS/ABBREVIATIONS

API	Application Programmer's Interface
COTS	Commercial Off-the-Shelf
DoD	US Department of Defense
DMSO	Defense Modeling and Simulation Office
Ec	Casualty Expectation
FED	Federation Execution Data
FOM	Federation Object Model
GIS	Geographic Information System
HCC	Human-Centered Computing
HCl	Hydrochloric Acid
HLA	High Level Architecture
IDL	Interface Definition Language
ILRO	Intelligent Launch and Range Operations
IT	Information Technologies
KSC	Kennedy Space Center
MOM	Management Object Model
NIST	National Institute of Standards and Technology
OMDT	Object Model Development Tool
RID	RTI Initialization Data
RTI	Runtime Infrastructure
SM	Spaceport Model

SOM	Simulation Object Model
SPEEDES	Synchronous Parallel Environment for Emulation and Discrete- Event Simulation
VR	Virtual Range
VTB	Virtual Test Bed
WES	Weather Expert System
XML	Extensible Markup Language

CHAPTER ONE: INTRODUCTION

Background of Study

Many real world systems and application areas of modeling and simulation are getting larger and more complicated. With the help of new science and information technology – especially the availability of powerful and accurate networked computers – engineers are now capable of building such complex systems and applications which were difficult to build even just one decade ago. Today’s real world systems generally consist of a large number of subsystems, are geographically dispersed over large distances and are operating in heterogeneous computing environments (Ghosh & Lee, 2000). Many such systems are often considered systems of systems.

A “System” is a group or combination of interrelated, interdependent, or interacting elements that form a collective entity. Elements may include physical, behavioral, or symbolic entities. Elements may interact physically, mathematically, and/or by exchange of information. Rouse defines complex systems as “systems whose perceived complicated behaviors can be attributed to one or more of the following characteristics: large numbers of elements, large numbers of relationships among elements, nonlinear and discontinuous relationships, and uncertain characteristics of elements and relationships” (Rouse, 2003).

Therefore the property of complex systems as a whole is nonlinear; hence the property cannot simply be derived from an integration of the properties of components of a system. However there are some common characteristics of complex systems: (1) they consist of a large number of interacting subsystems, (2) they exhibit emergence: that is, a consequence of the interactions between system components to achieve some objective which is difficult to anticipate from

the knowledge of the individual components, (3) their emergent evolution can be observed once the components have been integrated into a system or adapted into an environment where it is used. The appearance of emergent properties is the single most distinguishing feature of complex systems (Boccaro, 2004).

In engineering disciplines “a new system development is initiated either by user needs or new opportunity offered by advancing technology. The evolution of a particular new system from the time when a need for it is recognized and a feasible technical approach is identified, to the point in its development where it is introduced into operational use is referred to as the “system development process” (Kossiakoff & Sweet, 2003). Similar to system engineering, in software engineering there are a variety of software life cycle models in use. Since we are studying the development of simulation systems the emphasis is on the life cycle models in software engineering. Some of the well known models are the waterfall model, the rapid application development model, and the spiral model, among many others (Pressman, 2000; Vliet, 2000; Pfleeger, 2001; Sommerville, 2001; Thayer, Dorfman, & Christensen, 2002). All of these life cycle models are useful in developing a single system (or a single complex system) and most are useful in developing a system of systems with the exception of the traditional waterfall model. Because of the necessity for an understanding of the system level requirements at the beginning of the development process and because of the need for system level requirements to remain the same throughout the process, the waterfall model cannot be effectively used to develop a system of systems.

As presented earlier, the defining properties of complex system include (1) the *interactions* amongst interconnected components and/or the environment interacts in unanticipated ways and (2) the behavior of the overall system is different from the aggregate behavior of the parts

and knowledge of behavior of the components will not allow us to predict the behavior of the whole system, which is the property known as emergence (Sage & Olson, 2001).

Since many real world systems are systems of systems, the systems (subsystems) might be built in various disciplines, times, and often developers may not know of the existence of other systems (subsystems). Therefore the *interaction* cannot be defined in the development process of individual systems (subsystems).

The complex system of systems evolves the results of *interactions* with other systems and environments; and then adapts with its environment where it is used. In other words, when the configuration of the complex system changes such as when a system (subsystem) is added or disposed, or the new functionality of a system is required because of either new user requirements or advances in information technology, the systems (subsystems) need to interact with these changes and then adapt to them. However, the configuration change of the complex system, the future user needs, and the technology advance cannot be known in advance. As a result many software development life cycle models may not be applicable in the development of a system of systems.

The simulated complex system is usually decomposed to a level where subsystems or system components are individually defined and developed as functional simulation systems. In this dissertation we refer to a simulation of complex systems as an abstraction of the simulated complex systems in which systems (subsystems) interact with each other by exchanging the state of system information which changes either discrete in time or continuously, or a hybrid of both.

In this dissertation we propose a distributed simulation framework for the development of modeling and simulation of complex systems which are a system of systems, in which the defining properties of complex systems – interaction and emergence – are realized and the software

life cycle models (including prototyping) are used as metaphors, not actual development processes, to manage the complexity of modeling and simulation of the system. The framework is based on the interoperability of a simulation system enabled by the distributed simulation and the gateways which enable COTS simulation packages to interconnect to the distributed simulation engine.

Distributed Simulation

While Parallel/Distributed Discrete Event Simulation has been an active area of research for more than thirty years, researchers have until recently focused almost exclusively on fast execution of process and event oriented models of discrete event simulations. In the mid 1990's, High Level Architecture was initiated by the DMSO, US Department of Defense (DoD); the process has taken different aspects of Parallel/Distributed Discrete Event Simulation in order to support interoperability and reusability of existing simulation models developed at various times, purposes and organizations. The HLA is used as a distributed simulation engine in the VTB architecture for integration of both current and future simulation models. In general, the simulation languages/packages may have special areas of use, distinct advanced features, and require specific computing environments such as operating systems (OS), external application interfaces, and scripting languages. These characteristics of the modeling languages may impose difficulties when attempting to seamlessly integrate them with other simulation modeling languages/packages. We focus more on the VTB environment; in particular, with its interfaces among participating simulation models.

Since HLA was developed for reuse of the military simulation models, its main area of use has been military domain simulations. Although the HLA does not mandate the use of any specific software – it is designed to incorporate new technologies as they develop over time – currently the only supporting interfaces available constrain applicable program languages such as C++, Java, IDL and Ada. This is problematic when we interconnect simulation models in the VTB, many of which are developed by COTS simulation packages such as Arena, AnyLogic or SPEEDES. In addition the future models may need to be developed using one of the COTS simulation tools and the non-simulation (supporting) tools because these tools often offer rapid development cycle, some very specific advanced functionalities difficult to find elsewhere (e.g., Calpuff, ArcGis in VTB), and additional tools which are necessary in the process of model development, such as input/output analyzers, process optimization applications, visualization software of simulation execution and results. Many COTS simulation languages/packages do not expose their internal data structure and/or time advance mechanism to an external interface, both of which are required to interoperate with the HLA-RTI, and the interface programming languages need to be among the RTI supporting languages such as C++, Java or Ada.

To overcome the restrictions imposed by the HLA-RTI interface in use of COTS tools several approaches have been researched and implementation of these approaches has been reported. Some examples of such implementation that link COTS Simulation Packages to the HLA are Arena/ProModel (Charles & Frank, 2000), AnyLogic (Borshchev, Karpov, & Kharitonov, 2002), SLX (Strassburger, Schulze, Klein, & Henriksen, 1998; Strassburger, 1999), Matlab (Pawletta, Drewelow, & Pawletta, 2000), and MODSIM III (Johnson, 1999), among many others. In order to accomplish the objectives defined in the simulation system of the Virtual Test Bed (VTB) we have researched these approaches including a SPEEDES HLA gateway, the Distrib-

uted Manufacturing Simulation Adapter (DMS Adapter) with Arena, and the AnyLogic HLA support module HSM). These HLA interoperability approaches are discussed in detail in Chapter 3.

With each of these approaches comes a price. Although they provide a solution for the HLA interoperability, they sometimes cause compatibility issues and increase the complexity of simulation systems. For instance, the Manufacturing Simulation Adapter (DMS Adapter) provides a variant of Federation Object Model (FOM) in Extensible Markup Language (XML) format. The FOM written in XML format enables the simulation model to have extended data types, flexibility in individual document structure and format, ease of creation, parsing, interpretation, display by standard tools, and semantic validation of the file, among others. However when we are required to integrate the models written in simulation languages/packages which are using DMS Adapter and other HLA interoperability tools, the object classes or the interaction classes they are referenced may not be compatible in format. This requires the development of an additional component which translates data formats including data structure and semantics of attributes to make the FOMs compatible. The DMS Adapter also supports a subset of Application Program Interfaces (APIs) in the HLA Interface Specifications, which are sufficient for a distributed manufacturing simulation. The simplified DMS Adapter functions remove much of the complexity and many unnecessary APIs in some simulation projects. In some other application areas, the DMS Adapter interfaces that support a subset of the RTI APIs and the simplified time coordination in the DMS Adapter which implements “time stepped” synchronization approach may limit the range of interoperability enabled by the HLA.

Some simulation modeling languages support special features which may not be available in many other simulation modeling languages. In general, however, it is not easy to fully utilize

the functionalities that the simulation language supports. For example, SPEEDES is a unified parallel processing simulation framework that enables integration of objects distributed across multiple processors to speed-up simulation run. This feature enhances runtime, especially when exploiting the very large number of processors and the high-speed internal communications found in high performance computing platforms. It also supports multiple time management algorithms such as the sequential algorithm, time-driven algorithm, Time Warp algorithm, and Breathing Time Warp algorithm which is a combination of the Time Warp and Time Bucket algorithms. In optimistic time management, an event can be processed even if it may not be the next event to be processed in ascending time order while still maintaining repeatability and causality by using “rollback” techniques, whereas in conservative time management, an event will not be processed until it is known that there is no possibility of an event arriving in the past relative to the simulation time. Rollbacks restore state variables and retract events scheduled during the simulation time period that need to be rolled back. They also support the ability to roll an event forward without requiring large amounts of memory overhead if the state that the event depends on has not changed. Despite these advanced features, it is not easy for the modelers who have been using COTS simulation languages/packages for simulation modeling to learn the SPEEDES Modeling Framework and develop a distributed/parallel simulation model in SPEEDES.

Visualization

In addition to issues we presented, we have identified the importance of visualization of the interaction of the system components and the state of remote systems in distributed simula-

tion, especially in geographically distributed simulation systems like the Virtual Test Bed (VTB). Visualization as a part of a simulation system provides certain insights into the complex dynamics of the system that cannot be obtained using other analysis techniques. Visualization helps the modeler, the decision-maker, and non-technical persons to gain some understanding of the model being investigated. In the HLA-based distributed simulation, however, it is difficult, if not impossible, to provide the same level of insight to the user by the COTS visualization tools currently available. Mainly because the COTS visualization tools are integrated into its simulation engine or they are designed to support a stand-alone simulation execution instead of a distributed simulation run.

In a distributed simulation environment, although geographically dispersed simulation models may have their own visualization environments, it becomes difficult to provide a comprehensive, global presentation of a distributed simulation system. In order to support an effective decision-making process, informative visualization coupled with distributed simulation models are essential tools when dealing with a large and complex distributed simulation such as space shuttle operation models, supply chain simulations, or enterprise engineering models. Therefore, we believe that there is a clear need to have a visual representation of geographically distributed simulations on a single visual display in order to provide comprehensive insight of the entire distributed simulation, especially when decision making is the objective of the modeling.

While a standalone simulation model developing process usually takes advantage of the visualization tools embedded in COTS simulation packages, in the distributed simulation development process very few packages can be used to visualize the distributed simulations. There are two major reasons. First, the HLA framework doesn't take into account the visualization meth-

odology or the interface is too well tuned into the HLA framework. Second, although the HLA integrates various functional models or components, there is no general interface standard for COTS simulation packages in terms of communication and internal data presentation.

Case study: Spaceport simulation system in Virtual Test Bed Architecture

This dissertation is concerned with the integration of simulation models that are identified as the essential components of Spaceport and are facilitated with general-purpose discrete event simulation languages/packages based on distributed simulation engines such as High Level Architecture (HLA), and visualization of the participating simulation models in the distributed simulation.

The objective of the VTB is to provide a collaborative computing environment that supports the creation, execution, and reuse of simulations that are capable of integrating multidisciplinary models representing the elements of launch, ranges, and spaceport operations in order to assist with cost analysis, flow optimization, and other important decision making factors (Rabelo, 2002a). The VTB will provide multiple benefits, such as enabling risk management evaluations of existing and future vehicle frameworks, providing a technology pipeline for evaluating and implementing new solutions to existing problems, and enabling better knowledge management (Rabelo, 2002c).

Is Spaceport in VTB architecture a “complex system”? According to Barth,

“Spaceport technologies employ a life-cycle “system of systems” concept in which major spaceport systems -- launch vehicle processing systems, payload processing systems, landing and recovery systems, and range systems -- are designed concurrently with flight vehicle systems and flight crew systems. The result of applying this concurrent systems engineering approach will be robust space transportation systems for future

generations” (Barth, 2002).

A quick and easy answer can be found in (Shishko, Aster, & Cassingham, 1995).

“Most NASA systems are sufficiently complex that their components are subsystems, which must function in a coordinated way for the system to accomplish its goals. From the point of view of systems engineering, each subsystem is a system in its own right—that is, policies, requirements, objectives, and which costs are relevant are established at the next level up in the hierarchy. Spacecraft systems often have such subsystems as propulsion, attitude control, telecommunications, and power. In a large project, the subsystems are likely to be called ‘systems’”.

In addition our view of complexity in Spaceport simulation system is that since the spaceport system is a system of systems, each subsystem such as launch vehicle processing systems, range systems, etc is regarded as a component or a process in general concept of complex systems.

The simulation of Spaceport will be developed as a representation of these major spaceport systems. Therefore the simulation components may represent very distinct nature of the abstraction of the simulated system in heterogeneous computing environment to achieve the goal of the system; the nature includes the properties of interest, different time series (continuous, discrete, hybrid), target users, time of operation in the system, information it generates, etc. The output on a certain input to the simulation of Spaceport in which the subsystems of Spaceport are interplay based on its own dynamics of operation with multiple decision points will be far from linear.

In addition the Spaceport simulation system evolves and extends quickly with the environmental changes such as technology advances (IT, COTS tools, etc), new functionality needs such as “Mission to Mars, Moon”(NASA, 2004), the user requirement increases such as a model in higher resolution, etc. Therefore the defining property of complex systems, emergence, applies here. Over the life cycle of Spaceport simulation system the systems (subsystems) are intercon-

nected with themselves in a seamless way and an unambiguous way; and interact with and adapt to the environment. The future configuration of Spaceport simulation system by consequence of interaction and adaptation may not be deducible in advance especially in the stages in development of individual systems (subsystems). Therefore the integration of Spaceport simulation systems is an example of complex systems, and a system of systems as well.

Objective and Scope

The scope of this study could be described as: (1) analysis of Spaceport simulation system in VTB architecture with components of simulation models such as shuttle process, Virtual Range model, weather model, etc. (extensions to integration of additional simulation models can also be done in the future), (2) study software interfaces for integration of simulation models into the framework which can be used to integrate future models, (3) provide the visualization framework for distributed simulation system by utilizing an animation facility embedded in COTS simulation packages which traditionally do not provide animation functions for distributed simulation run. This framework can be used to visualize some simulation languages which do not have dedicated visualization capability such as SPEEDES, and (4) implement the framework for the case study of Spaceport distributed simulation system by using the HLA-RTI which could be used as a proof of concept of this study

The benefits of this framework are two fold. First, the framework supports the evolution of a simulation of complex system (that is, a system of systems) as operational requirements change. This is realized by applying the metaphor of the life cycle models especially the spiral model, most widely used model presently, which includes two variations termed incremental and

evolutionary is applicable throughout the system life cycle. By using these approaches in a system level, the system of systems evolves with requirements extended while the operability of the overall system kept. The fully functional simulation subsystems can be developed independently using different software languages/tools and then added to the system as requirement changes as in the incremental model approach; and improved or modified subsystems can be replaced with old one as technology advance and the user needs change as in the evolutionary model approach.

Second, the metaphor of the prototype model which is a well known variant of the waterfall model can be experimented within the operational system. With a gateway which enables a tool to interact within distributed simulation to COTS simulation package the experimental prototype simulation model can be built and then tested in relatively quick and with low cost. This is possible because the many COTS simulation packages/tools provide an integrated visual model development environment in which the user can build a model by “point-and-click” or “drag-and-drop” methods as well as often include pre-built modules (or templates) which help the modeler to build a quite large model in easy.

Dissertation Outline

This dissertation is structured as follows. Chapter 2 defines the context of this dissertation. The concept and architecture of the VTB are defined, and requirements and essential technologies in order to support the Spaceport simulation system are identified. Each of these technologies – discrete event simulation languages/packages, the HLA, and visualization framework required to implement the framework of distributed simulation for Spaceport simulation system in VTB – is reviewed extensively. Chapter 3 presents the HLA interoperability in COTS simula-

tion languages/packages. We detail the HLA interoperability approaches used in the simulation languages/packages that are selected based on the requirements for the Spaceport simulation system, outlined in Chapter 2. The discrepancies in the HLA interoperability employed in the selected simulation languages/packages with respect to communication and data compatibility are introduced. As one of our solutions for obstacles we introduced in terms of integration, we present the fundamental classes which are commonly provided in many COTS simulation packages for discrete event simulations. The classes are built upon the Process Model foundation in SPEEDES to promote the shared data compatibility between the models written on SPEEDES and those written in other simulation languages/packages. In Chapter 4 and 5, the cases studies (Sepulveda, Rabelo, Park, Gruber, & Martinez, 2004a; Sepulveda, Rabelo, Park, Riddick, & Peaden, 2004b) we have reported in Winter Simulation Conference in 2004 and the prototype VTB simulation system based on the concepts of integrated execution of different simulation models are presented with focus remaining on the HLA interoperability and the visualization of a remote federate in a distributed simulation. Finally, Chapter 6 concludes the dissertation and outlines possible direction for future Virtual Test Bed (VTB) research.

CHAPTER TWO: REVIEW OF RELATED TECHNOLOGIES

Introduction

This chapter presents an overview of the Virtual Test Bed (VTB), its architecture, and its requirements for the simulation system with a focus on integration of simulation models, each of which represent an operational element of a spaceport, range, or another similar system. The primitive discrete event simulation and its fundamental modeling elements are described briefly. Distributed/Parallel discrete event simulation concepts and High Level Architecture (HLA), the most state of the art distributed simulation engine currently available, are introduced. In (Swain, 2003) more than 40 COTS simulation languages/packages available at the time of survey are presented in detail. The modeling environments of several of these are presented, each having been selected based on the requirements for VTB and availability of the simulation language such as SPEEDES and simulation packages such as Arena and AnyLogic. Finally, the need for visualization in distributed simulation is researched.

Virtual Test Bed (VTB)

This section presents the concepts and the architecture of the VTB (Sepulveda et al., 2004a; Sepulveda et al., 2004b; Compton, Sepulveda, & Rabelo, 2003). NASA implemented the Intelligent Launch and Range Operations (ILRO) Program at Ames Research Center (ARC) to perform initial studies of a test bed with a demonstration (Bardina, 2001). The VTB Project is essentially an evolution of the ILRO test bed. “The objective of the VTB Project is to provide a collaborative computing environment to support simulation scenarios, reuse, and the integration

of multidisciplinary models that represent operational elements in ranges and spaceports. The VTB will provide several benefits, such as a risk management, evaluation of legacy and new vehicle framework, a technology pipeline, and a knowledge management enabler. The VTB will leverage current technological developments in intelligent databases from NASA ARC to present data and results as usable knowledge with associated security constraints and human-centered computing (HCC)” (Rabelo, 2002c).

According to (Barth, 2002), “Spaceport technologies must employ a lifecycle ‘system of systems’ concept in which major spaceport systems – launch vehicle processing systems, payload processing systems, landing and recovery systems, and range systems – are designed concurrently with the flight vehicle systems and flight crew systems.” One interesting characteristic of a complex system is that it is by default a system of systems which are themselves complex systems. To be faithful to concurrent engineering principles, we have to study the interactions among the different systems that are elements of the complex system. This system of systems is non-linear in nature and the interactions among the different components bring interesting emergent properties that are very difficult to visualize and/or study by using the traditional approach of decomposition. Therefore, the goal is to develop a VTB that can host models representing the systems and elements of a spaceport. These models will work together on the VTB in an integrated fashion, synthesizing into a holistic view and becoming a Virtual Spaceport. This Virtual Spaceport can be utilized to test new technologies, new operational processes, the impact of new space vehicles on the spaceport infrastructure, the supply chain, and the introduction of higher level decision making schemes. A Virtual Spaceport will allow for an intelligent visualization of the entire spaceport concept and the implementation of knowledge management strategies. The

central goal of the VTB project is to provide a virtual environment of the launch and range operations at Kennedy Space Center (KSC).

The VTB will integrate and adapt some of the current simulation models and bridge existing gaps to create a unique mission environment for the ILRO program. This realistic NASA mission environment will provide scientists within the NASA based Intelligent Systems (IS) project with a computing environment where they can implement schemes for high-performance human-automation systems. This integration will require the development of a computer architecture that allows for the integration of different models and simulation environments. The computing infrastructure will implement advanced ideas of integration, distributed and/or parallel computing, distributed simulation, security, and Web-enabled standard technologies such as Extensible Markup Language (XML).

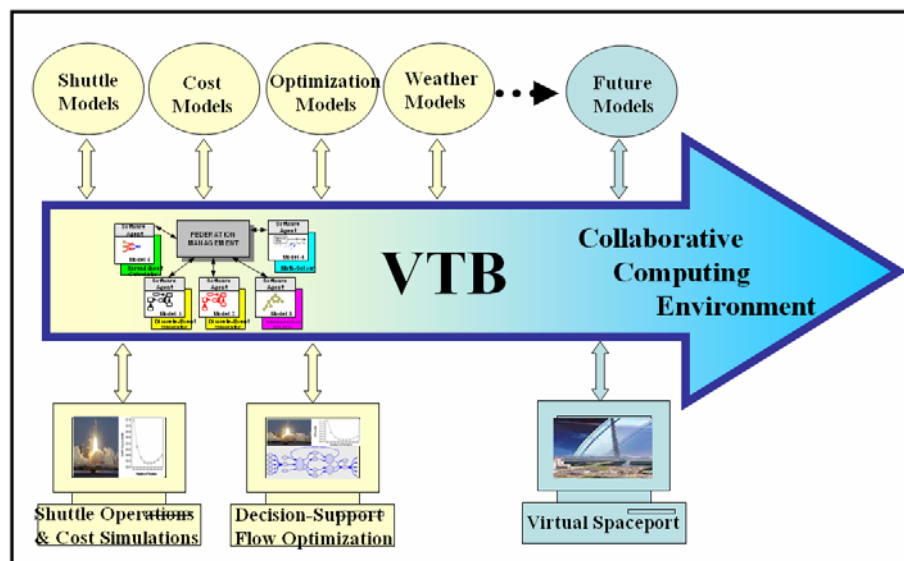


Figure 1 Concept of Virtual Test-Bed (adapted with modification from (Rabelo, 2002b))

Virtual Test Bed (VTB) Architecture

The VTB Architecture is composed of several sub-systems which include the Integration User Interface, Decision-Maker User Interface, Security Component, Integration System, Simulation System, the Model Functions Manager, the Model Library Manager, and Database System.

The Integration User Interface provides the capability to transfer models to the VTB. The user can integrate an existing model (and create extensions to it) using the tools and methodologies provided by this interface. The interface will have privilege mechanisms to provide a means of granting the VTB Model Integrator Expert access to the system (e.g., passwords, IP-based schemes of security) and the ability to perform security-relevant actions for a limited time and under a restrictive set of conditions, while still permitting tasks properly authorized by the VTB System Administrator. The Decision-Maker User Interface is the simulation interface which supports the development and execution of scenarios based on the models which have been integrated into the VTB. The Security Component provides password schemes, authentication, firewalls, Secure Socket Layer (SSL) implementation, maintenance and prevention mechanisms (e.g., virus protection and disaster recovery), certificates, and encryption (Rabelo, 2002a).

The Integration System takes the representation while the user interface supports the execution and together they develop the information outlined by the user (using the Integration User Interface) to formulate a hierarchical description of entities, activities, and interactions that is represented in an integrated model. The Simulation System will provide an environment to execute integrated simulators/models developed for specific elements of space operations into interactive simulator networks to support a single view of operations. For instance, NASA KSC has existing models that have been developed over time by different sources. These existing models

have been developed from different points of view and for different aspects of the operation cycle. Consequently, existing models represent different levels of resolution and have selected different representation methods for internal entities, activities and interactions. The Model Functions Manager provides the business logic for the various transactions to save the model configurations as specified by the Integration System. In addition to providing business logic, the Model Functions Manager also retrieves from the Database System the simulation models, data, and the configuration parameters needed by the Simulation System. The Model Library Manager will support the development and management (retrieval, saving, configuration management) of the library of models.

The Distributed simulation management system that controls the models and tools before and after execution of simulation and manages the models in the library of simulation components and tools will be a part of the VTB. These Distributed simulation management system capabilities will allow other platforms to be operated without extensive personnel management. Finally, the Database System stores the model and its details in a scheme appropriate for facilitating the operations of the Simulation System and the interface with NASA Ames Research Center ILRO VTB transactions to save the different model configurations as specified by the Integration System. The Model Functions Manager also retrieves the simulation models, data, and configuration parameters needed by the Simulation System from the Database System.

Simulation Data Models (SOM/FOM)	Simulation Execution / Integration Manager	Performance analysis/ Process Optimization	Model Library				
			Shuttle Process Model	Virtual Range	WES	Launch Control	Monte Carlo
Simulation Integration System							
Management Tools (XML, SQL,etc)	C++, Java, etc (SPEEDES, Silk, etc)		COTS simulation Packages (Arena, AnyLogic, etc)		COTS Tools (ArcGIS, Calpuff, etc)		
			Gateway, Adapter(wrapper)				
Database System	Distributed Simulation Systems (HLA-RTI)						
Operating Systems: Networking, Security, File Management System							

Figure 2 VTB System Architecture

Requirements for Virtual Test Bed (VTB) Simulation System

Global System Requirement: The VTB is created by one or more simulations and is a selective recreation of the real world. The simulated “world” consists of a representation of the environment, a well-defined set of objects that populate and evolve in that environment, and a communication mechanism to make sure that all interactions between the different elements occur in a managed and time consistent fashion. In order to accomplish the goals of the VTB framework we have identified several important aspects of the framework: (1) *Real-Time Visualization* allows (potentially widely distributed) users to collaborate using VTB, (2) *Knowledge and Information Repository* - a repository for storing data, software, object models and lessons learned, so that new exercises or scenarios or tests can be readily constructed, (3) *Integration Environment* - a suite of tools for integrating models, visualizing, planning, executing, collecting

data from, analyzing and reviewing scenarios, and (4) *Flexible and Evolving Architecture* - VTB will have the ability to flexibly reconfigure resources to meet new and changing needs.

Modeling Language Requirement: In recent years the improvement of functionalities of basic components in discrete event simulation and ease of use of simulation languages/packages has lead to the increased popularity and development of these modeling tools. A general-purpose simulation language provides flexibility in modeling, the availability to develop almost any requirements from a modeling environment, and the capability to generate efficient models with respect to execution speed. On the other hand, COTS simulation packages provide ready to use (built-in) features which cover a wide range of modeling necessity and usually have a highly functional user interface in modeling and modification (Law & Kelton, 2000). Weighing the advantages of these modeling languages/packages against the architecture and requirements of the VTB, the VTB team sought to find general-purpose discrete-event simulation languages and/or COTS simulation packages for the VTB Simulation System for use with both future and legacy technologies. The team identified some basic requirements for these simulation languages/packages which are as follows. (1) The languages/packages needed to support different hardware architectures ranging from a distributed network of fast workstations to a single computer. (2) In order to either develop the HLA-RTI interface or adapt the HLA-RTI interface into Virtual Test Bed (VTB) simulation system, C++, Java needed to be the main modeling language to support the development and implementation of the simulation. For the COTS simulation script language, however, Visual Basic Application (VBA) can be used to develop an advanced external interface from the model. (3) The simulation languages/packages must provide interfaces for developing external interfaces. These external interfaces may be used to control start/stop of model execution and develop the HLA interoperability and to allow scripting lan-

guages to be written as command messages on top of the simulation layered-architecture.

VTB is intended to provide a robust, flexible, easy-to-use architecture, which can incorporate current and evolving operational characteristics and scenarios to conduct investigations. Where COTS software products can meet task requirements safely, the COTS software is utilized instead of developing custom applications (Rabelo, 2002a). The software to be developed will be written in high-level languages such as Java, C, and C++, which have demonstrated a high degree of portability between platforms. This strategy provides a reliable system that is modular, expandable, and extensive. It is based on open hardware and software standards, easily incorporates new technology and user developed applications, and provides inherent user interface improvements.

High Level Architecture (HLA)

Parallel/distributed discrete event simulation refers to the execution of a single discrete event simulation program on a parallel computer, e.g., a supercomputer or a shared memory multiprocessor, or on a network of multiple computers (or processors). The primary reason for distributing the execution is to reduce the length of time required to execute the simulation or to enable larger simulations to be executed by utilizing resources from multiple computers when a single computer may not support enough computing resources to perform the simulation (Fujimoto, 2000; Fujimoto, 2003). As computer hardware technology advances and the cost of computing decreases, the application areas which can take a great advantage of this acceleration are limited only to the size of which are extremely large and/or execution time (in real-time or faster than real-time) critical applications.

However, other increasingly important aspects of Parallel/distributed discrete-event simulation technology such as interoperability, reuse of simulation components, and encapsulation of the modeling details of the simulation by separating network and model components are getting more attention from the simulation community. One example of such interest is the HLA, a standard distributed simulation interoperability architecture, developed by the Defense Modeling and Simulation Office (DMSO) of U.S. Department of Defense (DoD). The HLA has been developed based on the idea that no single monolithic simulation can support the needs of all users. An individual simulation or set of simulations developed for one purpose can be used in different distributed simulations by integrating these models as a component of the complex model (Judith, Richard, & Richard, 1998).

The HLA is a programming language-independent object-based distributed simulation architecture for promoting simulation reusability and interoperability by defining rules, methods, and data formats that simulation application must comply with. The High Level Architecture (HLA) was introduced by the Defense Modeling and Simulation Office (DMSO) of the Department of Defense (DoD) in 1996 and it was accepted as an IEEE standard for distributed simulation – IEEE 1516 – in September 2000.

The HLA defines a common high-level simulation architecture that supports the development of simulation applications by integrating other simulation components and tools such as visualization tools and real world systems. This architecture promotes interoperability and reusability of legacy simulation models in order to develop a new, complex simulation (Judith et al., 1998). Reuse of existing components may reduce the cost and time required to develop a new simulation.

The HLA defines terms used in the context of distributed simulation:

- *Federate*: a member of a federation; a federate refers to an actual simulation and the role in a distributed simulation is defined in its Simulation Object Model (SOM).
- *Federation*: a set of simulations (federates) interconnected through RTI; a Federation Object Model (FOM) and its supporting infrastructure, which is used to form a large model to achieve certain objective.

Major Components of HLA

The HLA comprises three major components: the HLA Rules, the HLA Interface Specification and the HLA Object Model Template (OMT), which describes the principles of the architecture and services required for supporting software to interface among simulations and the information model, respectively. The components describe the software architecture of the HLA as open instead of specifying a type of software development. Interoperability between federates is achieved by three major components: the HLA rules which describe federation and federate responsibilities, the Run Time Infrastructure which coordinates local simulation time managed by each federate with global simulation time in federation and controls the data transfer, and Object Model Template (OMT) which defines data structure, format of federates (SOM), and common information in federation (FOM). The following sections describe the three components in detail (Judith et al., 1998).

The HLA Rules

The HLA Rules define general information exchange principles required to ensure proper data transfer of objects (attributes) and interactions (parameters) between a federation and its federates, and describe the responsibilities of simulations and supporting tools participating in an HLA federation (DMSO, 1998e).

Federation Rules: (1) Federations shall have a FOM, documented in accordance with the OMT. (2) All representation of objects in the FOM shall be in the federate, not in the RTI. (3) During a federation execution, all exchange of FOM data among federates shall occur via the RTI. (4) During a federation execution, federates shall interact with the RTI in accordance with the HLA interface specification. (5) During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.

Federate Rules: (1) Federates shall have a SOM, documented in accordance with the OMT. (2) Federates shall be able to update and/or reflect any attributes of objects in their SOM, and send and/or receive SOM interactions externally, as specified in their SOM. (3) Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOM. (4) Federates shall be able to vary the conditions under which they provide updates to the attributes of objects, as specified in their SOM. (5) Federates shall be able to manage local time in a way which will allow them to coordinate data exchanges with other members of a federation.

The HLA Interface Specification and Run-Time Infrastructure (RTI)

The discussion in this section follows that of (DMSO, 1998c; DMSO, 1998b; Kuhl,

Weatherly, & Dahmann, 2000). The HLA Interface Specification defines the runtime services and interfaces to be used by federates and supports efficient information exchange between federates and the Run-Time Infrastructure (RTI) during a federation execution. It also defines the way these services are used in both their function and the Application Programmer's Interface (API). The services are classified as one of the six management groups of the FedExec life cycle: Federation Management, Declaration Management, Object Management, Ownership Management, Time Management, and Data Distribution Management. A high level illustration of the interplay between a federate and a federation is shown in Figure 3.

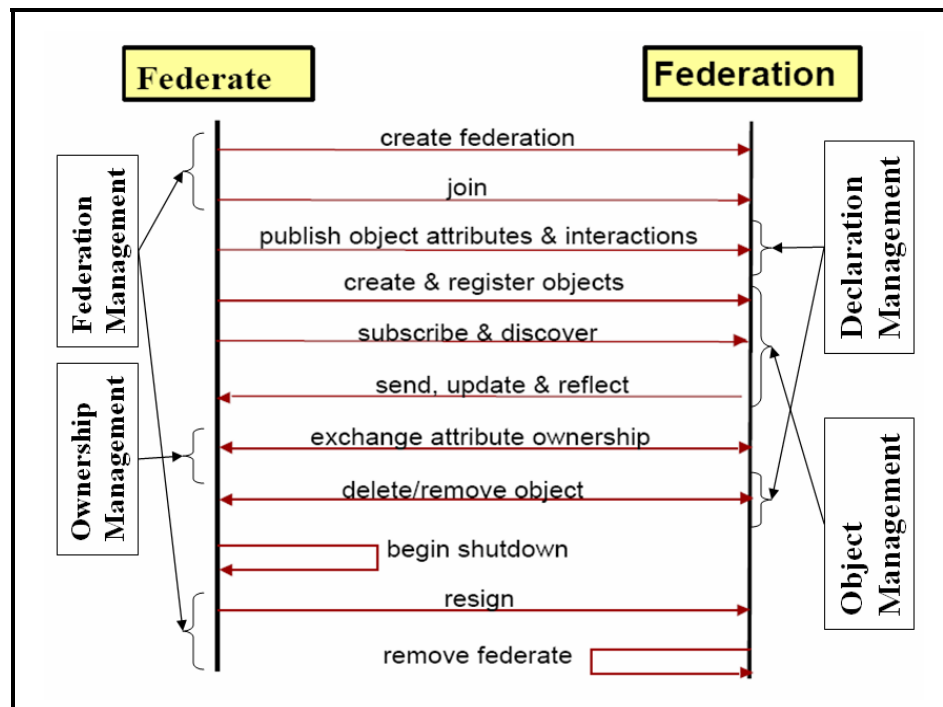


Figure 3 Federate – Federation Interplay.(adapted from (DMSO, 1998b))

The RTI, a software implementation of the HLA Interface Specification, defines the common interfaces for distributed simulation systems during the execution of an HLA simulation. It is the architectural foundation that promotes portability and interoperability. All shared information exchanged during a federation execution must be passed through the RTI. The RTI is

comprised of the following three components: the RTI Executive process (RtiExec), the Federation Executive process (FedExec), and the libRTI library. Figure 4 shows a configuration of federation with respect to the three RTI components.

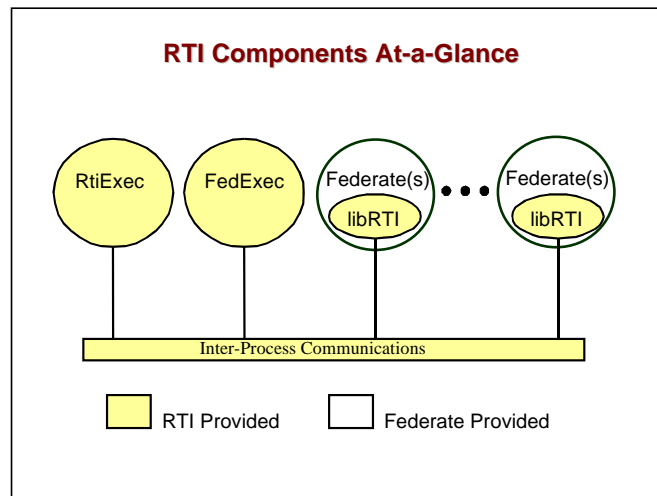


Figure 4 RTI Components At-a-Glance. (taken from (DMSO, 1998b))

The FedExec manages the process of joining federates and resigning the federation and facilitates data exchange between participating federates. A FedExec process is created by the RTI when the first federate successfully joins the federation and is eventually destroyed by the RTI when the last federate resigns from the federation.

The RtiExec manages the creation and destruction of multiple federation executions within a network. The RtiExec ensures each FedExec has a unique identification and directs the joining of federates to the appropriate federation. Although more than one federation can be running under a RtiExec, communication between federations is not possible.

The libRTI library extends RTI services to the federate developer. It enables the federate to access RTI services specified in the Interface Specification by RTIambassador and FederateAmbassador. Data exchange between federates in a federation occurs only through the RTI by the HLA rules and is accomplished by means of RTIambassador and FederateAmbassador. The

libRTI includes both the RTIambassador and the FederateAmbassador class. Passing information from a federate to the RTI is accomplished by calling services in the RTIambassador.

On the other hand, an event from the RTI to a federate and the response service subsequently requested by a federate are passed by asynchronously invoking FederateAmbassador “callback” functions that are implemented according to the function of simulation. Since FederateAmbassador is an abstract class, each federate must provide an implementation of the FederateAmbassador services. An instance of this federate supplied class is required to join a federation. The header file “RTI.hh” that accompanies libRTI includes declarations for class RTIambassador, the abstract class FederateAmbassador, and a variety of supporting declarations and definitions. The RTIambassador is implemented in libRTI and must be incorporated into each federate executable. Figure 5 shows the code responsibility of RTIambassador and FederateAmbassador.

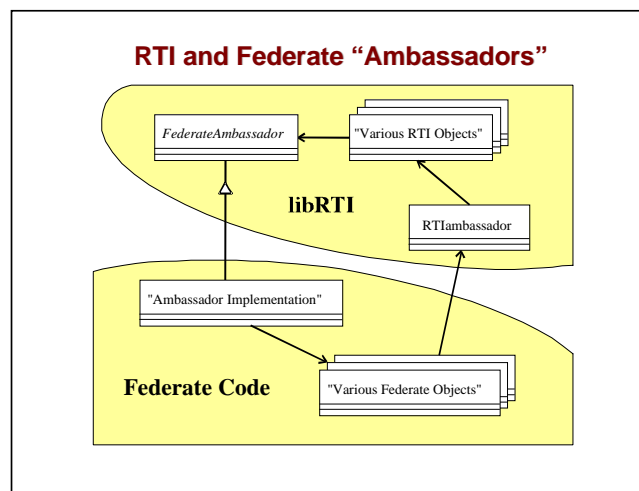


Figure 5 RTI and Federate Code Responsibilities. (taken from (DMSO, 1998b))

The HLA is a complex integration standard for a distributed simulation. The Interface Specification (DMSO, 1998c) includes over 150 different services for the RTIambassador interface and the FederateAmbassador interface. While the part of the HLA design that deals with

data sharing (publish and subscribe) is relatively straightforward, the overall architecture is complicated by the other supporting functions such as starting/stopping/saving/restoring federation, supporting different time management scheme, and transferring ownership of object attribute. To make the Interface Specification manageable, the specification is divided into six service management groups which are briefly described below.

Federation Management

The services in Federation Management group are mainly focusing on two types of operations. First, all basic functions related to such operations as defining a federation execution operation (creating federations, joining federates to federations, resigning federates from federations, and destroying federations). Second the federation-wide operations include the services controlling synchronization points, supporting state saves and restores. The four main functions in Federation Management service which are shown in Figure 3 are briefly described as following. (1) Creating Federation: A federation is created by calling the RTIambassador method `createFederationExecution()` which communicates the RtiExec process. If the specified federation does not exist, the RtiExec process creates a new FedExec process and associates it with the supplied federation name. If the specified federation already exists, a `FederationExecutionAlreadyExists` exception is raised, typically the exception is caught and ignored, and then the federate tries to join to the federation. (2) Joining to Federation: The `joinFederationExecution()` method is called to associate a federate with an existing federation execution. It requires the name of the calling federate, the name of the federation execution that the federate is attempting to join, and a pointer to an instance of a class implementing the `FederateAmbassador` callback functions. (3)

Resigning from Federation: The `resignFederationExecution()` removes the federate which calls the method from a federation. When a federate resigns from a federation, some additional actions related to update responsibility are passed to as an enumerated parameter. (4) Destroying federation: The `destroyFederationExecution()` method attempts to terminate an executing federation. If the invoking federate is not the last participating federate to terminate, a `FederatesCurrently-Joined` exception is raised, the federate can ignore the exception. (5) Federate Synchronization and Federation wide Save and Restore. The Federation Management includes the functions that allow federates to communicate explicit synchronization points for the time-ordered information exchanges as well as the services that support federation-wide saves and restores.

Declaration Management

Declaration Management services facilitate efficient information exchange by federates declaring their desire to generate and receive objects (with attributes) state information and interactions. Federates that produce objects (or a part of attributes) or that produce interactions must explicitly declare what they are be able to publish, and at the same way, federates that need updates of objects and interactions also must declare their interest in the attribute. Unlike object class, declarations of interaction must include all parameters. The classes and attributes used in the declarations must be consistent with the Federation Object Model (FOM). When a federate publishes information, the information is available to federation-wide. The RTI controls the distribution of information based on the federate interest declared by intention to subscribe so that the publishing federate generates object/interaction updates only if at least one subscribing federate exists. When a federate is no longer interested in any attributes of an object or interaction

class that were previously declared, the federate must declare the intention to stop publishing and/or subscribing.

Object Management

Object management includes services for registration, updates, and deletion of object instances for information production side and services for instance discovery and reflection on the parties interested in the object. To create or discover an object (or interaction) a federate must have published or subscribed that object class (or interaction class) through Declaration Management services. A federate also can delete an object instance, which in response, the subscribing federates will be notified and will delete the object instance. Object management also includes methods associated with sending and receiving interactions, controlling instance updates. Only a subscribing federate can request a value update of an object instance so that it can receive Reflect of a value update. The actual information exchange is supported by the Object management services.

Ownership Management

A federate is required to have ownership for an attribute of an instance before it can update. In HLA ownership simply means a responsibility of updating attribute values of an object instance. Ownership Management services provide dynamic transfer of ownership of object and attributes among federates. The RTI allows federates to share the responsibility for updating and deleting object instances. However only one federate can have update responsibility for an individual attribute of an object instance and privilege to delete an object instance at any given time.

Transfer of ownership can be initiated either by the current owner or the prospective owner. Only one federate that has the attribute "privilegeToDelete" for an object instance has the right to delete the object. Once the object is deleted all owners of attributes of the deleted object will be notified by the RTI that the object no longer exists. This will prevent publishing attribute values for the deleted object.

Time Management

Time Management deals with coordinating the exchange of events among federates in a federation. At the highest level, the federation appears to the RTI as a collection of federates that communicate by exchanging time-stamped events. Time Management services deal with the advancement and coordination of simulation time among federates in a federation. The time management services provide a variety of optional time management services for an orderly advancement of time during the execution. With default setting, the RTI does not attempt to coordinate time between federates which means federates are neither regulating nor constrained, so it is the federate designer's responsibility to select appropriate time coordination scheme among any combination of "regulating," and "constrained" depending on the purpose and the requirements of the federation. Regulating federates regulate the progress in time of federates that are designated as constrained.

Data Distribution Management

Both Declaration Management and Data Distribution Management support an efficient interest management mechanism for data exchange. The difference between the two is the level

of filtering. In Declaration Management, the RTI uses publication and subscription information in terms of object and interaction classes to control the update traffic. DDM provides a flexible and extensive mechanism for further isolating publication and subscription interests in terms of object instances and abstract routing spaces (Kuhl et al., 2000).

In DDM, a federation "routing space" is defined. The routing space is a collection of "dimensions." The dimensions are used to define "regions." Each region is defined in terms of a set of "extents." An extent is a bounded range defined across the dimensions of a routing space. It represents a volume in the multi-dimensional routing space (DMSO, 1998c; DMSO, 1998b).

The Object Model Template (OMT)

The OMT defines a common data structure and representation (format) of information for all objects and interactions exchanged between participating federates. The OMT enables interoperability and reuse of simulations and simulation components with respect to data modeling. Since the HLA does not adjust the contents and semantics of a FOM or SOM, a common documentation of shared information is required to support reuse of simulations (DMSO, 1998d).

Federation Object Model (FOM)

FOM describes all shared information as objects, object attributes, interactions and their parameters, which are essential to a particular federation. It does not contain actual information of instances of objects and interactions in the federate, but it provides a structure of objects and interactions.

Simulation Object Model (SOM)

SOM describes the objects, attributes, and interactions in a particular simulation which can be produced by the simulation and used by other federates in a federation. SOM describes the attributes of an object and parameters of interactions by type, cardinality, units and specification of update scheme.

Management Object Model (MOM)

MOM is an object model which defines a set of objects and interactions used to manage a federation. It is a standard part of FOM defined in the Interface Specification of HLA. It can, however, be extended by adding attributes or subclasses of an object, or adding interactions into the MOM. The RTI creates and manages instances of object defined in the MOM and updates attributes of it. The interactions defined in the MOM are used to manipulate the state of other federates and federation, e.g. adjust federation, request information, and report on federate activity (Kuhl et al., 2000; Fullford, 1999).

An Interface Specification prescribes the interface between each federate and the Runtime Infrastructure (RTI), which provides communication and coordination services to the federates. The RTI provides services to federates in a way that is analogous to how a distributed operating system provides services to its applications.

An Object Model Template (OMT) defines the way in which federations and federates have to be documented (using the Federation Object Model (FOM) and the Simulation Object Model (SOM), respectively). Federations can be viewed as a contract between federates where a common federation execution is going to be run. The HLA OMT provides a template for docu-

menting HLA-relevant information about federation objects (classes of simulation), their attributes (the data that describes the state of the objects in the federation), and the interactions that may occur between the objects in the federation.

A standardized structural framework (or template) for specifying HLA object models is an essential component of the HLA for the following reasons: (1) provides a commonly understood mechanism for specifying the exchange of public data and general coordination among members of a federation; it represents the format for a contract between members of a federation (federates) detailing the type of objects and interactions that will be supported across its multiple interoperating simulations, (2) provides a common, standardized mechanism for describing the capabilities of federation members; it represents a basis for comparisons of different simulations and federations, (3) facilitates the design and application of common tool sets for development of HLA object models.

Discrete-Event Simulation Languages and Packages

Modeling and simulation (M&S) is a powerful technology that helps to understand the dynamic nature of the existing or imaginary system being modeled. Discrete Event Simulation (DES) has especially been long recognized as an extremely valuable tool for analyzing complex systems. It provides a flexible tool capable of dealing with many design decisions that must be made before systems can become operational (Rogers & M.T.Flanagan, 1991). Traditionally the results produced by the simulation are used to identify the dynamic characteristics of the system by various methods of statistical analysis.

Discrete-event simulation refers to the modeling of a system in which the state of the system changes only at discrete points in time at which events occur. Events occur as a result of activity of entities and delays. There are a number of different approaches to discrete-event simulation: event, process, and activity approaches. In event approach a system is described by a set of events with related state changes at the time of each event. In activity approach a system is modeled by identifying areas where a number of events are grouped in order to describe an activity carried out by an entity. In process approach a system is described by the following process: “a time-ordered sequence of interrelated events separated by intervals of time, which describes the entire experience of an *entity* as it flows through a *system*”(Law et al., 2000; Pidd, 1998). A discrete event simulation can be built by either COTS simulation package or a general-purpose simulation language. Following sections introduce COTS simulation packages as Arena and AnyLogic and SPEEDES, a general-purpose simulation language. These modeling tools are selected based on availability, the number of existing models written in the modeling tool which we are going to integrate into the VTB, and the requirements of the VTB simulation system.

Synchronous Parallel Environment for Emulation and Discrete-Event Simulation

Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES) is a general-purpose discrete-event distributed simulation engine and modeling framework for building complex and interoperable parallel/distributed simulations in C++. It was developed at the Jet Propulsion Laboratory by Dr. Jeff Steinman (Bailey, McGraw, Steinman, & Wong, 2001; Metron, 2003; Steinman & Wong, 2003). SPEEDES provides a parallel processing framework that enables integration of objects distributed across multiple processors to get simu-

lation speed-up. This feature enhances runtime, especially when exploiting the very large number of processors and the high-speed internal communications found in high performance computing platforms.

SPEEDES supports multiple time management algorithms such as the sequential algorithm, time-driven algorithm, Time Warp algorithm, and Breathing Time Warp algorithm which is a combination of the Time Warp and Time Bucket algorithms. In optimistic time management, an event can be processed even if it may not be the next event to be processed in ascending time order while maintaining repeatability and causality by using “rollback” techniques, and in conservative time management, an event will not be processed until it is known that there is no possibility of an event arriving in the past relative to the simulation time. Rollbacks restore state variables and retract events scheduled during the simulation time period needed to be rolled-back. They also support the ability to roll an event forward without requiring large amounts of memory overhead if the state that the event depends upon has not changed. This is known as lazy event re-evaluation: Breathing Time Warp algorithm (Bailey et al., 2001; Steinman, 1998a; Steinman, 1990; Steinman et al., 2003).

The SPEEDES architecture provides communications, event, time management, and a modeling framework. We are focusing on the modeling framework. The SPEEDES modeling framework is comprised of four fundamental components to provide the basic functionalities needed for event-based simulation modeling: (1) object manager, (2) simulation object, (3) events, and (4) messages (Fullford, 1999; Bailey et al., 2001). Figure 6 shows the components and hierarchy of the SPEEDES Modeling Framework.

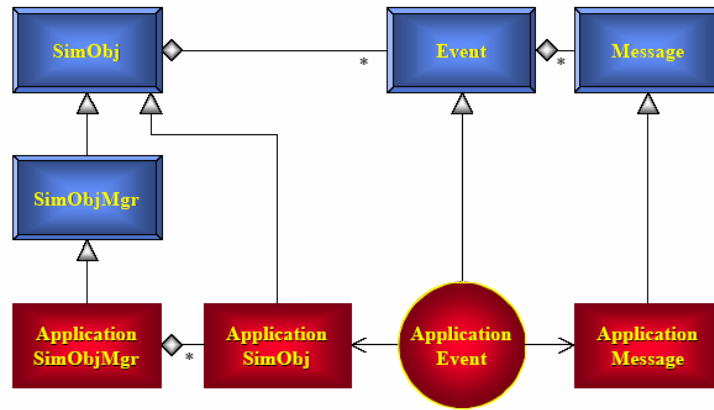


Figure 6 SPEEDES Modeling Framework (taken from (Bailey et al., 2001))

Object Manager

When the simulation is initialized one simulation object manager for each simulation object type is created on each node. Object manager controls the creation, initialization and destruction of simulation objects and the decomposition of said objects which is a function of placing simulation objects to nodes. Decomposition of objects can be done by automatic methods (block, scatter) or user-defined manner. Block decomposition distributes the simulation objects to nodes evenly. Scatter decomposition distributes the simulation objects such that simulation objects with consecutive kind IDs are located on different consecutive nodes. SPEEDES also supports file-driven user specified decomposition in which the user must provide placement of simulation objects and nodes in *SimObjPlacement.par* file (Metron, 2003).

Simulation Object

Simulation objects are the fundamental concept behind SPEEDES Modeling Framework which represent entities in the simulation system. It consists of a set of attributes which maintain

the state of the object and the methods which define the activities of object. The type of attributes may be primitive base types from C++ or rollbackable types if the attribute is state sensitive.

Simulation object class in SPEEDES provides the primitive functions to schedule event, process event handler, and response to interactions. All user objects must inherit either from the simulation object class (SpSimObj) or from one of its subclasses. Figure 7 shows an example of

S_Shuttle simulation Object structure (Metron, 2003).

```

1 #ifndef S_SHUTTLE_H
2 #define S_SHUTTLE_H
3
4 #include "SpSimObj.H"
5 #include "SpDefineEvent.H"
6 #include "SpDefineSimObj.H"
7 #include "RandomGenerator.H"
8
9 class S_Shuttle : public SpSimObj{
10 private:
11     UniformRand myRandom;
12     RB_int state;
13     RB_double timeOfLastStateChange;
14 public:
15     S_Shuttle();
16     virtual ~S_Shuttle();
17     void Init();
18     void Sieze();
19     void Release();
20     virtual void Terminate(double simTime);
21     static int GetNum(){return 3;}
22 };
23
24 DEFINE_SIMOBJ(S_Shuttle, S_Shuttle::GetNum(), SCATTER);
25 DEFINE_SIMOBJ_EVENT_0_ARG(Shuttle_Sieze, S_Shuttle, Sieze);
26 DEFINE_SIMOBJ_EVENT_0_ARG(Shuttle_Release, S_Shuttle, Release);
27
28 void PlugInShuttle()
29 {
30     PLUG_IN_SIMOBJ(S_Shuttle);
31     PLUG_IN_EVENT(Shuttle_Sieze);
32     PLUG_IN_EVENT(Shuttle_Release);
33 }
34
35 #endif

```

Figure 7 S_Shuttle Simulation Object

The S_Shuttle class inherits from SpSimObj and it includes the typical C++ constructor and destructor along with two virtual methods Init() and Terminate(). It is highly recommended, however, that in SPEEDES Modeling Framework simulation objects perform the necessary processes in Init() and Terminate() that are typically in C++ class placed in constructor and destructor, respectively. DEFINE_SIMOBJ (in line 24) macro is used to create a simulation object

manager for the simulation object, S_Shuttle. During the simulation initialization, the object manager will then create the user-specified number of simulation objects. `DEFINE_SIMOBJ_EVENT_0_ARG` (in line 25) macro is used to create an event, Shuttle_Seize, when it is call method `Seize()` is to be executed.

Events

Simulation object events are a part of a simulation object and are used to change the values of the state variables in simulation objects. They are defined as a public method, the most accessible level, so that any simulation object in the simulation may schedule the simulation object events. SPEEDES provides a set of macros that turn methods on simulation objects into events, plugs these events into the SPEEDES framework, and generates functions for scheduling these events. To make scheduling events convenient, the macros automatically build a global function for each event defined, which users can use to invoke each event (Metron, 2003).

A simulation object event can be created by `DEFINE_SIMOBJ_EVENT` macro (as shown in line 25 and 26 in Figure 7) and `PLUG_IN_EVENT` macro (as shown in line 31 and 32 in Figure 7) which register the event into the SPEEDES framework. This simulation object event will be executed when an object calls a schedule function and also a scheduled event can be canceled when the event scheduled in the future needed to be changed or canceled.

In addition to simulation object event, SPEEDES provides Local Events and Autonomous Events. An object may define Local Events on its sub-objects to manage the sub-object with self-scheduled events. Since Local Events are defined on sub-components of an object, the accessibility of that event is limited to its simulation object. Unlike Simulation object events and Local

events, autonomous events are separate from the Simulation object which they act. Users can create an Autonomous Event object which inherits from SpEvent class and define a method on the event which will be scheduled by a simulation object. Autonomous Events are often used with the lazy option which allows users to rollforward a rolled back event to prevent re-execution of the event if re-execution of the event does not change the outcome of the simulation.

Message

When an event schedules a new event, a message is created by SPEEDES with header information that defines the type of event, simulation time, type of simulation object and its local Id. The header information is used to create a corresponding event object by the destination node. SPEEDES provides another way to schedule and process events. Users can define methods in their simulation objects to be invoked as events. Applications can schedule these event methods using a type-checked event-scheduling interface provided by SPEEDES.

Among others, SPEEDES also provides the following modeling facilities. (1) SPEEDES provides HLA-RTI interoperability in two ways. First, SPEEDES provides a gateway between a SPEEDES-based simulation model and a RTI so that a SPEEDES-based federate can be developed without integrating Local RTI Component (LRC) of HLA. Second, SPEEDES has implemented the HLA RTI interface so that the SPEEDES kernel itself can serve as an RTI. Under this scheme, multiple SPEEDES and/or non-SPEEDES federates operating on high-performance computing platforms can interoperate via the standard RTI interface, with RTI communications implemented by high-speed shared memory mechanisms (this version of SPEEDES is not available to us). (2) SPEEDES modeling framework is an Object-Oriented architecture, and therefore

has a significant impact on the development of simulations. Individual classes can represent entities in a system. Such a representation, in turn, facilitates the distribution of the simulation models on different processors and the design of parallel simulation experiments. As a distributed discrete event simulation framework, it allows distribution of various objects over multiple processors and coordinates the simulation activities among various objects that are distributed.

(3) SPEEDES provides interfaces for developing external modules. These modules provide functionalities that allow interoperability between various simulation systems and tools that will be able to interact with the simulation model while it is running. This interface enables the users to control time advance of the simulation, receive information about the simulation state, and invoke events in the simulation. Hence, an external module can be implemented as a graphical display of simulation or a user interface to the simulation. (4) SPEEDES provides an advanced feature called Load Balancing. This feature enables the user to balance the objects that require more processing on a faster processor, leading to improvements in run time performance. (5) A parser was integrated into its framework so that a parameter file could be used for setting initial values for simulation objects or making run-time changes to the simulation. (6) SPEEDES also provides some diagnostic tools including event tracing and event usage statistics (Bailey et al., 2001; Hanna & Hillman, 2002; Metron, 2003; Steinman et al., 1999; Steinman et al., 2003).

Arena

The Arena simulation modeling package is a visual modeling environment. A model comprises model logic, animation and model definitions. The package also includes several tools such as input data distribution-fitting, output analysis, debugging, and optimization tool. Many

activities in a model building process can be done using Modules and Templates by Drag-and-Drop and visual coding. The modules and templates support hierarchical modeling. In addition to the basic model logic, similar to that of a flowchart, advanced features can be modeled using external programming languages such as SIMAN, VBA, or C. The results of simulation run can be stored in Microsoft Access by default and viewed in Crystal Reports, also the external input/output data can be linked to files in such formats as text, spread sheet, XML, and other ADO (Bapat & Sturrock, 2003).

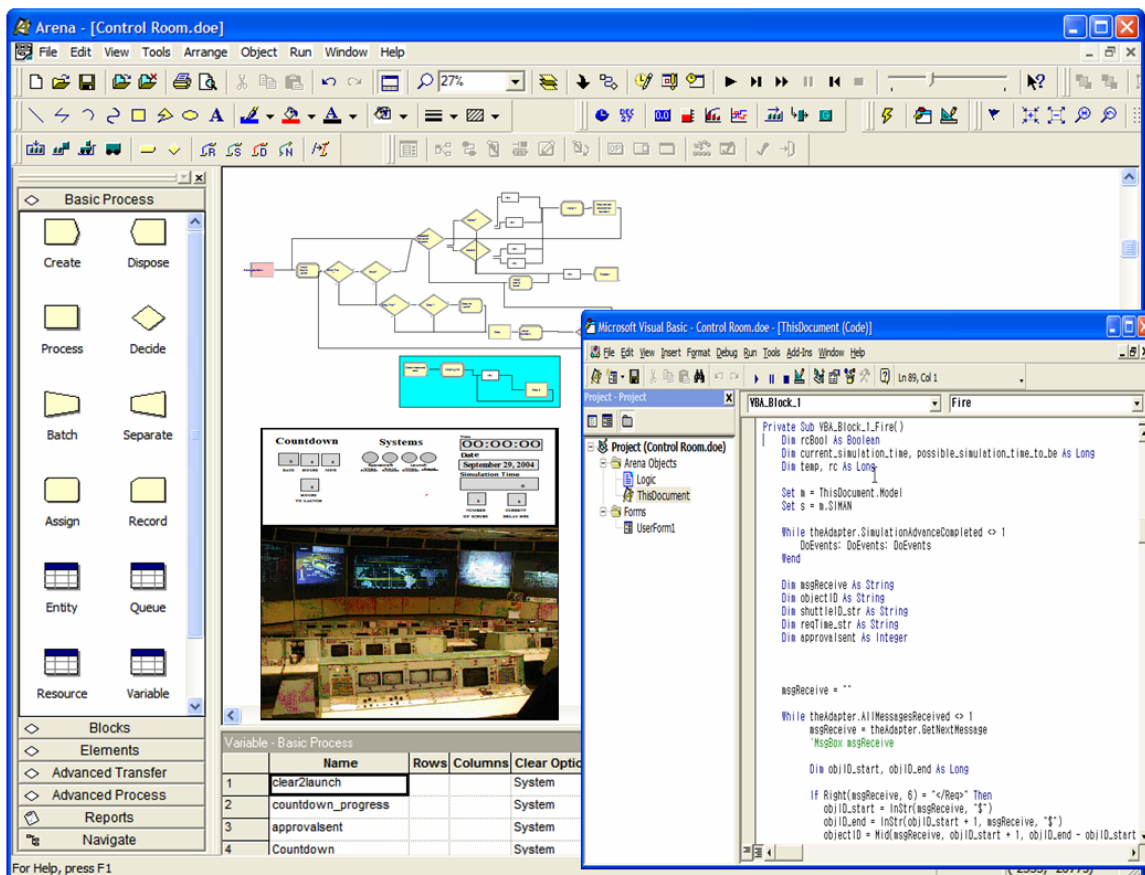


Figure 8 Arena Modeling Environment

The Arena modeling environment provides an embedded Visual Basic development Environment with which an interface can be developed for a model to interact with external applications such as Excel, VBA, Visio, Access, DDE/OLE Automation supporting applications.

AnyLogic

AnyLogic is a Windows-based, general-purpose simulation environment for complex discrete, continuous, and hybrid systems developed by XJ Technologies. It includes graphical model Editor, data collection and analysis facilities, debugging and visualization tools, and a Code Generator which converts the model into Java codes. The modeling language of AnyLogic uses UML-RT (UML for Real Time) – collaboration diagrams and statechart diagrams – to model hierarchical object-oriented models and specify behaviors of objects. It can be executed on any Java platform over Hybrid Engine and also supports interoperability to HLA-RTI (Borshchev et al., 2002). Figure 9 shows the architecture of AnyLogic Modeling and Simulation Environment

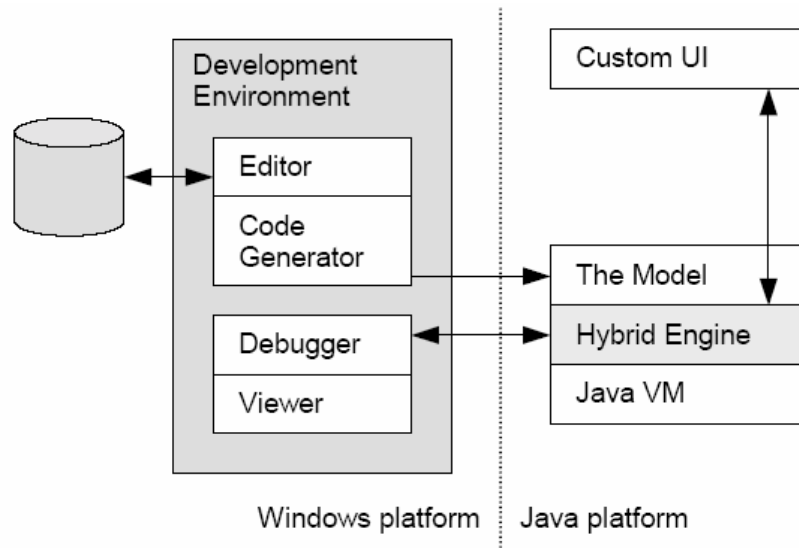


Figure 9 Architecture of AnyLogic Modeling and Simulation Environment (taken from (Borshchev et al., 2002))

Visualization of Distributed Simulation Systems

In the simulation field, from the inception of modeling and simulation study in 1970s, the visualization of a simulation system (or animation) has been established as an essential component of simulation study. Along with the statistical analysis, visualization of a simulation system is used to help the simulation developer and the end user by supporting heightened understanding and discussion of the model. Visualization is “an interface between two powerful information processing systems—the human mind and the modern computer. Visualization is the process of transforming data, information, and knowledge into visual form making use of humans’ natural visual capabilities. With effective visual interfaces we can interact with large volumes of data rapidly and effectively to discover hidden characteristics, patterns, and trends.” (Nahum, Stephen, & Stuart, 1998) There have been significant efforts to integrate visualization capabilities with general purpose simulation languages and packages. Many currently available commercial simulation packages include a wide range of animation tools capable of high resolution settings and

utilizing good 2D/3D authoring tools (Peter Lorenz, 2003). In recent years, along with the advent of high-performance, low-cost graphics technology, the quality and the realism of animation has rapidly advanced and proved visualization to be an extremely useful tool for modeling and simulation.

The trend makes it “hard to conceive of a simulation not using visualization techniques in some form.”(Steven D.Farr & Alex F.Sisti, 1994) Visualization has become a critical component of simulation technology. Today we can’t imagine doing a simulation without some kind of visualization to help communicate results and obtain a better understanding of a model’s behavior." (Rohrer, 2000) Furthermore, several authors point out that use of visualization could result in an increased acceptance of simulation results and be the element that determines the success of the project (Blocher, 2002; Nahum et al., 1998; Rohrer, 2000; Steven D.Farr et al., 1994). Although a stand-alone DES has been successfully applied to many engineering domain applications and used to address a wide range of complexity problems, many believe it is difficult for a single simulation model to provide for all of the user's requirements and predict all the features it would need. It is thought that Distributed Simulations could be the solution for this limitation by providing interoperability among simulation model components; the complex problem can be constructed by interconnecting many sub-components and user's future needs can be added to the current model without major changes.

Use of Visualization in Simulation

Visualization of a simulation provides an understanding of the complex dynamics of a system that are otherwise impossible to obtain by using conventional analysis techniques. The

following are the areas for which visualization can be incorporated and evaluated as an extremely valuable tool for both the model developers and the end-users (Law et al., 2000; Rohrer, 2000; Swider, Bauer, Jr., & Schuppe, 1994; Steven D.Farr et al., 1994).

Visualization is a highly effective means of communicating the essence of a simulation model to decision makers and upper management who may not have the technical knowledge required to understand the statistical outcome of a simulation; it may also promote communication among the project team. It is a reliable method of presenting concepts of model dynamics to the end-users who may not be aware of the technical details of the model. Presenting a visualization of the system being investigated could save a great deal of time by eliminating the lengthy presentation of statistical analysis needed “to be presented, explained, justified, and questioned” (Rohrer, 2000) of the system behavior.

Verification is the simulation modeling process of comparing the conceptual model with a computer representation of the conceptual model, while validation is the process of determining whether the output performance measures from the model match up to those of reality. In general, insuring the accuracy of the model and confirming model validity can be difficult without using animation. Since animation provides visual trace of events at the place where the events are relevant, it is a helpful tool in uncovering modeling error when the event happened as opposed to having to wait until the simulation run ends. In addition, visualization is useful in identifying a sudden but short interval of surging in some model variables, which is not easy to identify through average statistics collected at the end of the simulation. Visualization is a tool to verify the correctness of a model. Steven claims that “the most widely used technique for establishing conceptual model validity is “face validation”, which involves having domain experts view the animated behavior to determine whether it “reasonably” captures the essence of the

problem.” (Steven D.Farr et al., 1994) In some cases where the system being modeled does not exist or is modeling an operational behavior on an alternative structure of an existing system, validation becomes more complicated. If that is the case, visualization is critical in order to present how the proposed system works or how these two systems – the current system and the alternative one – perform in terms of the measure of interest.

Visualization is an essential component of training simulation. It is an interface between the simulation and the trainees. It provides not only the current state of simulation but also the effects of their response. Stafford claimed the following as a benefit of visualization in training simulation: “With simulation, operators get a global view of the impact made on other departments when making operational decision. This is not possible in a real operation because operators have a very localized view of the entire facility.” Farr and Sisti believe that visualization “allows testing of systems and techniques, and training of operational personnel, where testing with the real world system or environment is impossible, infeasible or costly.”(Stafford, 1995; Steven D.Farr et al., 1994) Animation has proved to be a useful tool in assisting engineering analysis of simulation systems, often leading to improvements in system design or operational procedures. When animation is used as an analysis tool, it helps users explain events such as simulation bottlenecks, conflicts, and deadlocks.

Although visualization helps model developers with many modeling tasks, it is no substitute for statistical analyses of model output. In addition, it takes time to build realistic animation scenes. Although the adaptation and incorporation of animation capabilities into a simulation is not without cost, the benefits far outweigh the expense and effort. In summary, a successful simulation project should be a combination of a sufficient statistical analysis and well designed animation.

Visualization in Distributed Simulation

Visualization helps the modeler, the decision-maker, and non-technical people to gain an understanding of the model being investigated. However in the HLA-based distributed simulation, it is difficult, if not impossible, to provide the same level of insight to the user by the visualization tools currently available. Although geographically dispersed simulation models have their own visualization environments, it becomes difficult to provide a comprehensive presentation on a global view of distributed simulation system. In order to support an effective decision-making process, an informative visualization coupled with distributed simulation models could be essential tools for large and complex distributed simulation; such as space shuttle processing operation models, supply chain simulations or enterprise engineering models. Therefore, we believe that there is a clear need to have a visual representation of geographically distributed simulations on a single visual display in order to provide comprehensive insight of whole distributed simulation, especially when the objective of modeling is a decision making purpose.

While a standalone simulation model developing process takes advantage of these visualization tools embedded in COTS simulation packages, in the distributed simulation development process, very few packages can be used to visualize the distributed simulations. There are two major reasons for this lack of distributed simulation visualizations. First, the HLA framework doesn't take into account the visualization methodology or possess an interface that is well tuned into the HLA framework. Second, although the HLA integrates various functional models or components, there is no general interface which interconnects the wide range of those models.

In the HLA-RTI based distributed simulation, since the HLA provides a common interface architecture to the simulation components, it is easy to disseminate the state of a system to

the federation in textual format. In order to visualize the textual information, a dedicated animation application as a display federate must be developed.

In VTB we have developed a primitive visualization for distributed simulation by incorporating the COTS simulation package, AnyLogic, to make possible the functional and logical visualization of important systems, and allow engineers to more thoroughly investigate and display the operational processes of the simulation which is located in the remote site.

CHAPTER THREE: THE HLA INTEROPERABILITY IN SIMULATION LANGUAGES/PACKAGES

During the last three decades a variety of Commercial Off-The-Shelf (COTS) simulation tools have been developed and used widely in many areas of the industry. Despite the main purpose of the HLA which was to provide interoperability to military applications to promote reuse of existing models and tools, recent trends show that its use has been spread to a wide range of other domains including academia and industry. With this trend, there have been number of approaches reported which enable communication and data exchange between COTS Simulation Packages and the HLA-RTI through an interface or a toolset for COTS Simulation Packages in the form of either modifying COTS Simulation Package's framework or developing a wrapper/gateway (or middleware) without significant change in the framework. Some examples of such implementation that link COTS Simulation Packages to the HLA are Arena/ProModel (Charles et al., 2000), AnyLogic (Borshchev et al., 2002), SLX (Strassburger et al., 1998; Strassburger, 1999), Matlab (Pawletta et al., 2000), and MODSIM III (Johnson, 1999) among many others.

It is thought that by incorporating various COTS simulation tools which are typically specialized in a certain area with the HLA interoperability, the range of a federation (or distributed simulation) can cover is broad by reusing existing models (or components) built into the specialized COTS tools.

Federate Requirements to Become a HLA Compliant

In order to support the HLA Interoperability for COTS simulation packages, there are two types of requirements – (1) one derived from the HLA Interface Specification of Ambassador paradigm, (2) and the other derived from distributed simulation. Strassburger also presents that the four general approaches that make a model compliant to the HLA are (1) modifying modeling framework, (2) changing the model source code independently with the tool, (3) developing an external programming interface such as Windows Dynamic Link Library (DLL) or Component Object Model (COM), and (4) coupling by a gateway program. The following section describes the requirements for a federate (or COTS simulation model) to become HLA compliant (Strassburger et al., 1998; Strassburger, 2001; Boer & Verbraeck, 2003).

Federate Requirements in the HLA Specification

As discussed in Chapter 2, a federate interacts with various simulations only through RTI by the HLA rules and it is accomplished with the RTIAmbassador and FederateAmbassador classes. Both the RTIAmbassador and FederateAmbassador class are a part of libRTI. While the federate code provides the internal functionality of the simulation, the local RTI Components (LRC) provide the RTI services specified in the Interface Specifications through the RTIAmbassador class and assist the federate in communicating with the RtiExec and the FedExec. Since the FederateAmbassador class is abstract, each federate must implement the callback methods in the FederateAmbassador class. All requests from a federate to RTI are accomplished by calling the RTIAmbassador method call. On the other hand, an event from the RTI to a federate and the subsequent response of service requested by a federate are passed by asynchronously invoking Fed-

erateAmbassador “callback” functions implemented according to the function of simulation.

Figure 10 shows the components of a federate.

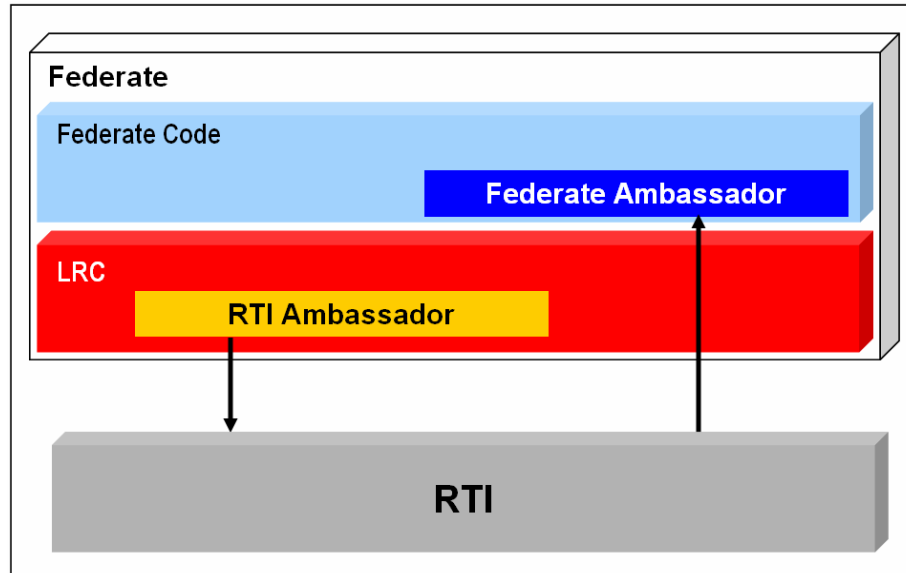


Figure 10 RTI and Federate “Ambassadors” (DMSO, 1998a)

In order to make a COTS simulation package compliant to the HLA, a federate is responsible for invoking proper APIs included in RTIambassador which is provided as a Dynamic Link Library (DLL). In addition, a federate must implement a set of “callback” functions which are provided as a form of C++ header file by HLA. The functions can be called by RTI as an asynchronous response of a request from the federate.

There are two methods for making a simulation model that is compliant to HLA. In both cases, an interface must prescribe to the same requirements discussed in section 3.2. First, if a general-purpose high-level simulation language is used to develop a model, the method is straightforward, the federate code directly includes the libRTI library and implements the FederateAmbassador abstract class definition which are provided in the form of header (“.hh”) files when using C++. Second, in the event that a COTS simulation package is used, since many COTS simulation packages currently available do not support direct call to C++ libraries as an

external interface, it must provide one of the alternative approaches which support the requirements for a federate. In addition to the interface to libRTI, it also requires access to some internal data that is needed to connect to other simulation models (Boer et al., 2003). The following section describes some solutions for COTS simulation packages such as Arena and AnyLogic, and SPEEDES, a general-purpose high-level simulation language.

HLA Support in Modeling Languages and Packages

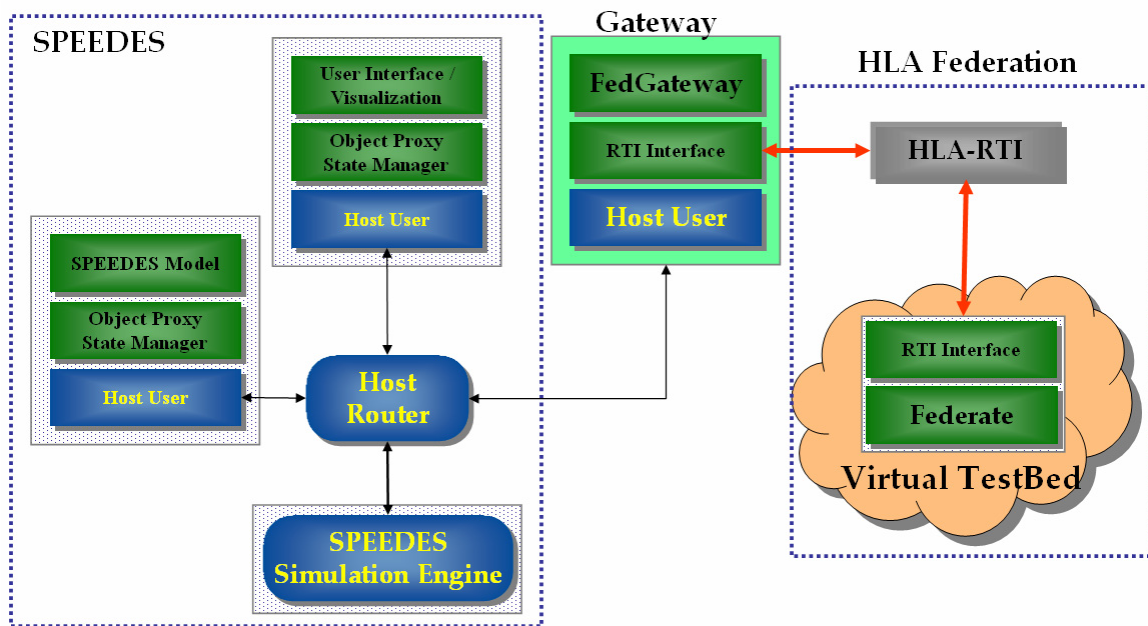
This section presents three different approaches for the simulation languages and packages have been used in the VTB simulation system. Each of these modeling tools applied the different approaches introduced in Chapter 3. The major factor in selecting one approach and the coverage of the RTI services by the approach may depend on availability of source code for the modeling framework and the environment the federation that is being used. In general, the RTI services in Federation Management, Declaration Management, and Object Management are an essential part of a federate, but those in Ownership Management, Time Management, and Data Distribution Management may not be required in many situations.

SPEEDES

We have been using SPEEDES as one of the discrete-event simulation languages for our HLA implementation because it meets the requirements listed in Chapter 2. SPEEDES is a software framework based on NASA-patented algorithms for building parallel C++ simulations. SPEEDES allocates events over multiple processors to get simulation speed-up. This feature enhances runtime, especially when exploiting the very large number of processors and the high-

speed internal communications found in high performance computing platforms. It connects a model to the HLA through the SPEEDES HLA-gateway. An HLA gateway provides a direct interface to the RTI. The federate gateway is implemented as an entity, which means that it can work with the entire Federation Object (FO) and interaction system in SPEEDES. By subscribing to all FOs and Interactions with both SPEEDES and with the RTI, the gateway is able to coordinate two-way flow of information. FOs that are created by an entity are discovered by the gateway and then registered with the RTI. In a similar manner, FOs that are discovered from the RTI are created and published by the gateway. Subscribing entities will then discover the FOs through the SPEEDES interest management system. The gateway is implemented through the use of FOs (FO classes, or S_HLA class). FOs provide applications with an automated framework for SPEEDES to distribute the exportable attributes of a SimObj to (1) subscribing SimObjs within a SPEEDES based federate, (2) external modules, or (3) other federates within an HLA federation. Each FO provides a well documented set of exportable attributes that collectively characterize the public state of an application's Simulation Object Model (SOM) (Bailey et al., 2001; Steinman et al., 2003; Steinman, 1998b).

FO attributes are normally declared within SimObjs as exportable state variables that are mapped to FOs through pointer references. FO attributes are then used as normal data types in application code. Through operator overloading, FO attributes detect when modifications are made. Events are automatically scheduled to reflect the changes of these attributes to all subscribers. Figure 11 shows an integration of a SPEEDES model to the HLA federation through the HLA-gateway.



HLA Gateway is a SPEEDES external module that handles communication to/from the RTI. It joins itself to the federation and then, on behalf of the SPEEDES simulation, performs all the services required as a federate including join/resign from the federation, publish/subscribe interaction classes, time management, and route interactions to/from the federation. Therefore a SPEEDES simulation and the HLA Gateway together work as a general federate. The HLA gateway uses the “*gateway.par*” parameter file to customize a distributed simulation environment, which includes parameters for the RTI (federate name, federation name and .fed file) and the federation (lookahead, federation synchronization point, etc). The HLA gateway also uses the “*conversions.par*” parameter file to convert objects (and attributes) and interaction (and parameters) between a SPEEDES user model and the HLA Gateway. The parameter file includes a list of objects and interactions that it either intends to publish or subscribes to from the federation. Each of the objects and interactions must have a corresponding “*SPEEDES name*”, “*RTI class*

name”, and a designator (PUBLISH or SUBSCRIBE). Figure 12 shows the format of “*conversions.par*” parameter file.

INTERACTIONS {		
<i>// SPEEDES Interaction</i>	<i>RTI Interaction</i>	<i>PUBLISH / SUBSCRIBE</i>
Request_for_Launch	Request_from_shuttle	PUBLISH {
Pub_request_ID	Request_ID	
Pub_request_time	Request_Time	
Pub_request_shuttleID	Request_ShuttleID	
}		
Approval_from_Control	Approval_to_shuttle	SUBSCRIBE {
Sub_Approval_ID	Approval_ID	
Sub_Approval_LaunchTime	Approval_LaunchTime	
Sub_Approval_shuttleID	Approval_shuttleID	
}		

Figure 12 An example of "conversions.par"

A SPEEDES simulation sends out HLA interactions simply by scheduling SPEEDES interactions in the usual way. This works by having a special object inside the simulation subscribe to the SPEEDES interaction classes, receive interactions and pass them along to the gateway. The gateway translates the SPEEDES interaction (and parameters) to the corresponding interaction (and parameters) defined in FOM and sends it out to the federation.

Arena with Distributed Manufacturing Simulation (DMS) Adapter

The Distributed Manufacturing Simulation (DMS) Adapter is a component of an HLA-based infrastructure for distributed simulation of manufacturing facilities. The adapter was developed by the National Institute of Standards and Technology (NIST) as part of the MISSION project: an international, collaborative project and part of the international Intelligent Manufacturing Systems (IMS) Program (McLean & Riddick, 2000b; McLean & Riddick, 2000a).

The DMS Adapter's infrastructure was designed to support the integration of multiple manufacturing simulations both with one another and with other manufacturing software applications. The DMS Adapter facilitates the adoption of distributed simulation in manufacturing environments by providing an interface that reduces the complexity of integrating simulations using the HLA to a level that is practical for manufacturing simulations. This is supported by several architectural design goals. (1) It reduces HLA interface complexity. In the DMS Adapter, the methods (APIs) for Federate Ambassador (callback) and RTI are grouped and it exposes only about 35 methods. The methods can be divided into three management groups: Simulation Execution group, Message Management group, and Object Management group (NIST, 2001). Table 1 shows the three management groups and their interface methods (2) Since the DMS Adapter that includes an implementation of federate ambassador is provided in form of Component Object Model (COM), legacy simulations are not required to implement Federate Ambassador. (3) Data from the RTI delivered asynchronously are stored in the internal storage of the adapter associated with a federate and the stored data will be passed to the federate upon request. This sequence of data exchange enables use of procedural languages such as Visual Basic Application (VBA) and other similar scripting languages. (4) While the representation of common objects and associated attributes in a federation are defined in the FOM, the instances of these objects are maintained by the federate. Moreover the internal representation of these objects may differ from simulation to simulation. To address the problem of having to develop a FOM for each federation, the details of object class and the associated attributes, as well as the interactions and associated parameters, are not defined in the FOM, instead a generic object class and a generic interaction class are defined. The generic object class contains an XML string that describes the structure and semantics of the object. (5) Finally, the DMS Adapter supports a "time-stepped" synchronization ap-

proach. When a simulation wishes to advance to a certain simulation time, it checks the global simulation time of the federation and then requests to advance. The methods for these processes are provided by the Adapter (McLean et al., 2000a; McLean et al., 2000b).

Table 1
The DMS Adapter Interface Methods

Adapter Methods		
Time advancement and Simulation Execution	Message Management	Object Management
<ul style="list-style-type: none"> • Initialize • Terminate • AdvanceSimulation • SimulationAdvanceCompleted • GetExecutionState • GetSimulationTime • GetProperty • SetProperty 	<ul style="list-style-type: none"> • GetNextMessage • AllMessagesReceived • SendMessage 	<ul style="list-style-type: none"> • CreateObject • UpdateObject • DeleteObject • GetObject • GetObjectValue • SeizeObject • ReleaseObject • SelectObjects

While the DMS Adapter minimizes the changes needed for simulations to participate in a federation, it provides mechanisms to coordinate the time between legacy simulations, facilitate message exchange, and provide facilities for object creation, update, storage, deletion, and transfer of ownership (McLean et al., 2000a). To realize the mechanisms each instance of the adapter maintains internal repositories for several kinds of information, the adapter's "internal data". Some of the internal data maintained by each instance of the adapter follows: federate member list, time management data, local/remote object cache, incoming/outgoing message queue, adapter properties, and subscription and filtering data. The only way to access the adapter's internal data is through the adapter's methods. A conceptual view of the DMS adapter architecture is shown in Figure 13.

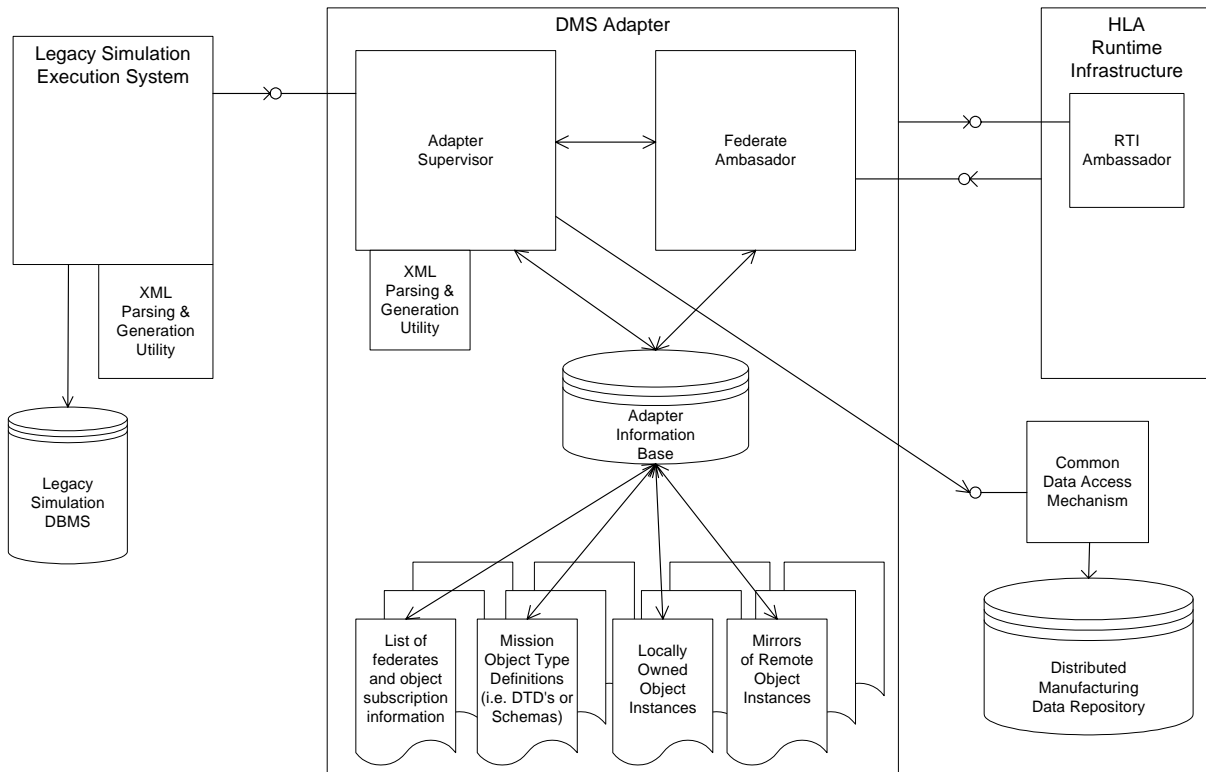


Figure 13 Initial Adapter Architecture(taken from (Kuhl & Riddick, 2000))

In addition to the HLA operability, the adapter enables the user to tune the properties of the simulation through “*GetProperty*” and “*SetProperty*”, and/or by an initialization file in XML format. The properties of the adapter can be customized are *Initial Simulation Time*, *SimulationStepSize*, *SimulationName*, *FederationName*, *DebugMode*, among others.

In the DMS Adapter architecture, Extensible Markup Language (XML) documents are used to specify an “initialization file” and to describe Objects and Messages. XML is a Meta language that describes data in plain text format. It is created as a way to structure, store and interchange information independent of hardware, software, and application. The advantages of using XML documents as input for some of the adapter methods are (McLean et al., 2000b): (1) the model designer can create, view, and edit the documents using the standard XML tools (Document Object Model (DOM), Document Type Definition (DTD), and XML Schema, etc), which

are independent of the simulation modeling packages, (2) the XML standard technique such as eXtensible Sytle Language (XSL), XML Path Language (Xpath) can be used to access all or parts of objects and attribute values as a part of the architecture, e.g., *GetObjectValue*, (3) due to the advantage of XML's hierarchical document structure and extensibility, the object structure in the model can be built with the same internal structure of the actual object using the same vocabulary which makes clear understating of data. Figure 14 shows an example of an initialization file in XML format.

```

<InitializationFile>
  <Properties>
    <InitialSimulationTime>0</InitialSimulationTime>
    <SimulationStepSize>10</SimulationStepSize>
    <SimulationName>MonteCarloSim</SimulationName>
    <FederationName>VirtualTestBed</FederationName>
    <Notifications>Enabled</Notifications>
    <DebugMode>Enabled</DebugMode>
    <UseManagerMode>Disabled</UseManagerMode>
    ...
  </Properties>
</InitializationFile>

```

Figure 14 An Example of Initialization File

Objects (and attributes that characterize the object) and interactions (and parameters) are stored in the form of XML documents. The header information which is used for controlling objects and interactions is stored as attributes of the top level document element – *ObjectType*, *MessageType* - and the attributes are stored as elements of the *ObjectType*. Figure 15 shows Objects and Attributes, a general Message, and Notification Message in XML format.

<i>Objects</i>
<Accident ObjectID="101" OwnerID="MontecarloSim" UpdateCounter="3" LastUpdateTime="1205" TransferTo ="VirtualRange"> <Time>025</Time> <Location> <X>539.134</X> <Y>3146.924</Y> <Z>01958.66</Z> </Location> <Concentration> <HCl>0.154E+07</HCl> </Concentration> </Accident >
<i>Message</i>
<LaunchDecision MsgID="3" Timestamp="1205" Sender="MissionControl" Recipient="LaunchPad"> Approved </LaunchDecision>
<i>Notification Message</i>
<Notification Type="FederateJoined" Name="VirtualRange" ID="5"/>

Figure 15 Object and Message Format

HLA Support Module (HSM) for AnyLogic™

AnyLogic™ is a Windows-based, general-purpose simulation environment for complex discrete, continuous and hybrid systems developed by XJ Technologies. It includes a graphical model Editor, data collection and analysis facilities, debugging and visualization tools, and a Code Generator which converts the model into Java codes. The modeling language of AnyLogic™ is UML-RT (UML for Real Time) – collaboration diagrams and statechart diagrams – to model hierarchical object-oriented models and specify behaviors of objects. It can be executed on any Java platform over Hybrid Engine and also supports interoperability to HLA-RTI.

Originally AnyLogic modeling framework is so called “closed-architecture” which means that the engine interfaces are not available for the external modules to control the process

of model execution. To support HLA interoperability for AnyLogic modeling framework an add-on package, HLA Support Module (HSM), was developed by XJ Technologies to facilitate interaction between the AnyLogic kernel and the RTI. This interface enables AnyLogic to support a wide range of RTI services in the following service groups: Federation Management, Declaration Management, Object Management, and Time Management. HSM uses the stepHook interface which puts a hook on simulation engine performing model time steps. The StepHook interface uses *nextEvent(double)* methods which is executed just before each time step to schedule a next time step and *timeStepDone()* method which is called by the engine right after the system clock has been adjusted to the time given by *nextEvent(double)* method. The StepHook interface enables AnyLogic models to exchange messages and synchronize Local Simulation Time to federation Global Time with the federation (Borshchev et al., 2002). The structure of HLA federation with AnyLogic federate is shown in Figure 16.

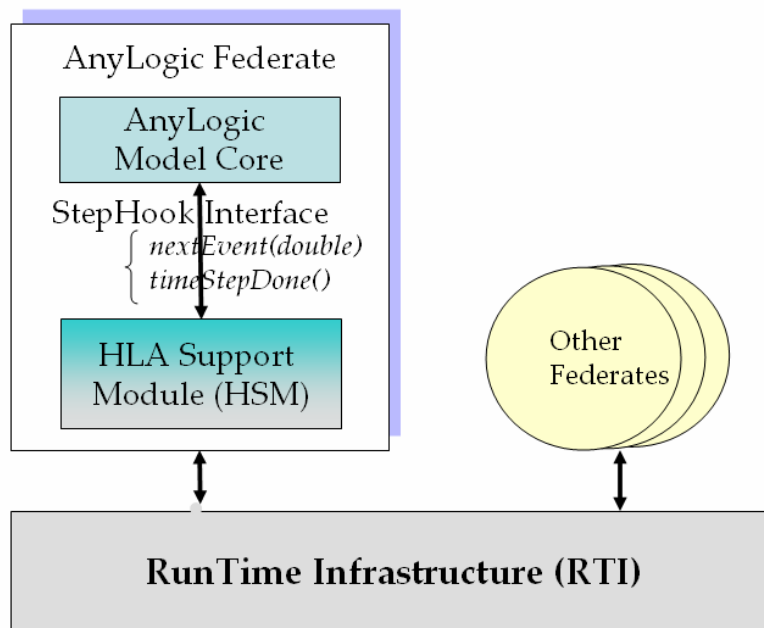


Figure 16 Integrating HLA and AnyLogic (adapted from (Borshchev et al., 2002))

The HLA interoperability in AnyLogic is supported by an HLA Support Module (HSM) class library. The HLA Support Module (HSM) enables a federate to invoke the RTI services by using the following classes: (1) supporting most of low-level RTI services (HLAHelpers), (2) publishing and subscribing objects and interactions (HLAObjectClass/HLAInteractionClass), (3) sending and receiving objects and interactions (HLAObject/HLAInteraction). The HLA Support Module (HSM) also presents two port classes which queue receiving objects and interactions as HLAObjectUpdatePort and HLAInteractionTranceiverPort, respectively. The AnyLogic's powerful visual modeling environment coupled with the HLA interoperability may enable the user to develop component simulations (federates) and development of distributed simulations (federations) using easy-to-use, user friendly, drag-and-drop graphic modeling environment. This modeling environment also can be used as a prototype development tool to provide possible solutions and validate the approaches and then the approved method can be reimplemented in the real environment. This will reduce the time and cost in federation development and avoid many errors on early phases of the development process (Borshchev et al., 2002).

Implementation of The Basic Discrete Event Simulation Class in The SPEEDES Process Model

Law and Kelton pointed out that one of the most important decisions in a simulation study is the selection of software (or simulation language/package). It is desirable that the software should neither be too difficult to use nor not be flexible enough (Law et al., 2000).

One of the decisions we have made concerns SPEEDES; despite SPEEDES providing exceptional functionalities including parallel execution, multiple time management, load balancing, and external module interface, among others, it requires profound knowledge in not only

simulation techniques but also SPEEDES modeling framework and its general-purpose programming language, in this case C++. All this makes steep learning curve for the modelers who have experience in using simulation packages but not in programming languages.

In order to address this, we have defined a set of SPEEDES classes using Process Model on top of the SPEEDES modeling framework, e.g., Simulation, Entity, Process, Resource and Queue, etc. These components are commonly used in COTS simulation packages. The design of the basic component classes is apparent for two reasons.

First, the set of pre-defined classes may simplify the modeling process and remove the needs for in depth knowledge of SPEEDES in the area of industrial simulation modeling. Second, our focus on this matter is to implement an interface with which other simulation models in VTB (AnyLogic, Arena, etc.) can interact with a SPEEDES model not only by means of communication but also by data sharing. The means for communication is provided by SPEEDES gateway interface to HLA, and the language to talk to each other will be provided by the basic modeling classes which are analogues to the building blocks provided as modules or templates with data format specified in OMT. We overview the SPEEDES modeling framework (SMF) with a special focus on Process Model, and then present the fundamental modeling classes in SPEEDES Process Model.

Process Model

Unlike event-based simulation, in which the activity is divided into independent event routines that describe the state changes by logical consequences of an event, process-based model defines a process as “a time-ordered sequence of interrelated events separated by intervals

of time, which describes the entire experience of an entity as it flows through a system” (Law et al., 2000). Process-based modeling paradigm is more common than other approaches (event-based or activity-based approach) in modern modeling tools. This is because writing a process-based simulation is often simpler, more intuitive, and easier to maintain than writing the same simulation in other paradigms. Whitehurst and Brutocao believed that “Object-oriented modeling coupled with process-based simulation support provides an environment in which real-world systems and simulated systems enjoy a high degree of fidelity. The fidelity between the system being analyzed and the program being developed reduces development time and produces better software quality”(Whitehurst & Brutocao, 1998) .

Whitehurst has implemented the process model using discrete event primitives, which is a set of macros that warp the SPEEDES code required to implement the semantics of process.

In SPEEDES, a process is a point-to-point event that uses special macros that allow for both exiting code execution at any point and reentering the code at the exit point at some later simulated time without losing local variable state or algorithmic context. The process model in SPEEDES defines a set of APIs which include Initializers, Wait Reentry Points, Semaphore Reentry Points, and Ask Reentry Points. The process model requires a minimal process block-structure for a process: P_VAR, P_LV, P_BEGIN, and P_END which mark a start of process model, defines process model local variables, the beginning of process model user code, and the end of process model user code, respectively. Wait Reentry Points support WAIT and WAIT_UNTIL constructs; WAIT waits the specified amount of time while WAIT_UNTIL waits until the specified simulation time has been reached before process model continuation. Semaphore Reentry Points support WAIT_FOR and WAIT_FOR_RESOURCE constructs which work with semaphore classes. These constructs break out of their waits based on the setting of

semaphore or after the specified wake-up time has expired. The semaphore classes used for the constructs are SpLogicalSem, SpCounterSem, and SpResourceSem. These semaphores allow for sharing logical variable, non-negative variable, and integer/double variable types as resources across multiple simulation objects. Ask Reentry Points construct enables users to send/receive data to/from another simulation object method. To employ the ASK in a process, the user must first use a macro that converts a method into an event that can be used by the process model.

Simulation Modeling Classes

In general building a simulation model using a modeling language like SPEEDES requires a profound knowledge of many simulation techniques as well as the modeling framework of the language, both of which are not often possessed by model developers. Therefore it is desirable to have conceptual model building blocks that are commonly found in many COTS discrete event simulation packages. This may help to speed the development time of project, prevent logical programming error, and increase interoperability with other COTS simulation packages with respect to information sharing.

To provide the basic building blocks to the modelers, several simulation modeling classes which are identified as basic components of discrete event simulation in (Arief & Speirs, 2000; Braude, 1998; Law et al., 2000; Rossetti, Aylor, Jacoby, Prorock, & White, 2000) have been implemented. We believe that these class greatly simplify the programming process representation of the system being modeled in the VTB environment based upon SPEEDES modeling framework. The classes are S_Simulation, S_Entity, S_Process, S_Resource, and S_Queue. These classes are derived from the SimObj and S_SpHLA classes implemented in Process Model mac-

ros in SPEEDES. The static properties of the classes are depicted by the UML class diagram seen in Figure 17. In addition, Figure 18 shows the dynamic aspects of a simulation system in a UML sequence diagram. It consists of objects and their relationships including messages that might be sent from one object to another by scheduling simulation object events.

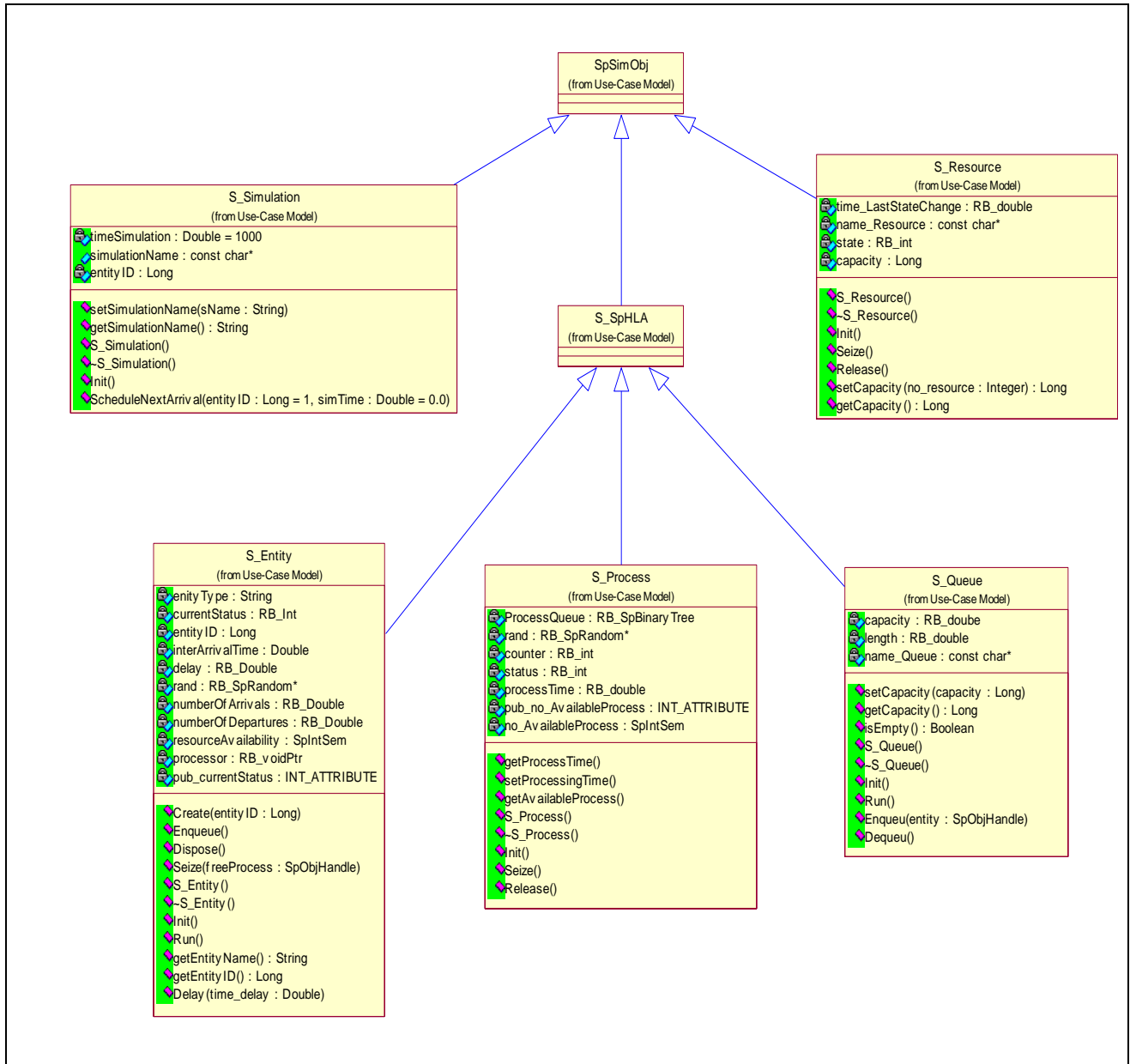


Figure 17 Class Diagram of Simulation Classes

S_Simulation class

S_Simulation class is derived from SpSimObj class. S_Simulation class contains the general properties related to the simulation and provides: (1) number of arrivals (batch size) at each arrival and more importantly, the event that schedules dynamic arrival of entities. The first entity arrival is scheduled at a specified time in this class as a model parameter and additional entity arrivals will be scheduled by the function of inter-arrival time defined in the S_Entity class by calling “CreateNextEntity” event in S_Simulation class.

S_Entity class

S_Entity class is derived from S_SpHLA class. S_Entity class represents an active object in the simulated system. Instances of the class differ by name, attributes, activities, and interactions with other objects by scheduling events during the simulation. S_Entity class contains the general properties of the active simulation object in the simulation: (1) setting inter-arrival time of the same type of entity, (2) obtaining its status, and (3) a series of activities that represent the flow of entity through the simulated system is modeled in Process() simulation event. Within the Process() event each activity, including the scheduling of the next arrival of entity, requesting a resource by joining to a queue, process, and departure are defined as simulation object event: Create(), Seize(), and Dispose() respectively. The model designer usually creates a model by implementing an active entity (S_Entity or sub class of E_Entity) which flow through the system in which the entity changes the state of the system and that of itself by interacting with other entities. It also registers attributes that are defined in *Objects.par* file to proxy system. For each at-

tribute macro `DEFINE_ATTRIBUTE` is called with a pair of arguments with attribute name in the class and correspond to the attribute listed in the `Objects.par` definition.

`S_Process` class

`S_Process` class is derived from `S_SpHLA` class. `S_Process` class controls the status of the object of `S_Resource` class. The entity schedules the `Seize()` event on `S_Process` with the number of resources required to start a service. The waiting for resources, `seize()`, and `release()` events in `S_Process` class are handled by the process model loop which repeats the process of waiting for resources, assigning resources to the entity, and releasing resources from the entity. By separating `S_Process` class and `S_Resource` class, the resource can be simulated not only as a group (number) but also an individual object in that way states of each resource can be modeled such as schedule of resource, failure.

`S_Resource` class

Resource in discrete-event simulation is a system component that provides a service to the entities. Typically the capacity of resources is limited, hence entities compete each other for service from resources to perform activities assigned to and it may result in a delay. `S_Resource` represents passive objects which are used by `S_Entity` object and managed by `S_Process` class objects in the activity. When more than one `S_Resource` object are requested by an entity the `S_Process` class object are responsible to allocate the resource object and free it after finishing the activity. The state of an `S_Resource` object may be active (seized, released), failure, inactive.

S_Queue class

Queue is a modeling element that refers to a place entities waiting for a service from the process along with resources when the resources are not available to the entity because either all the resource are allocated to other entities or there are not enough resources to start the process.

S_Queue class is derived from S_SpHLA class. The S_Entity interacts with the S_Queue so that new S_Entity class object can be added to the S_Queue. The S_Queue also interacts with S_Process so that new entities can proceed with their activities (get services from S_Process object) if the S_Process is available.

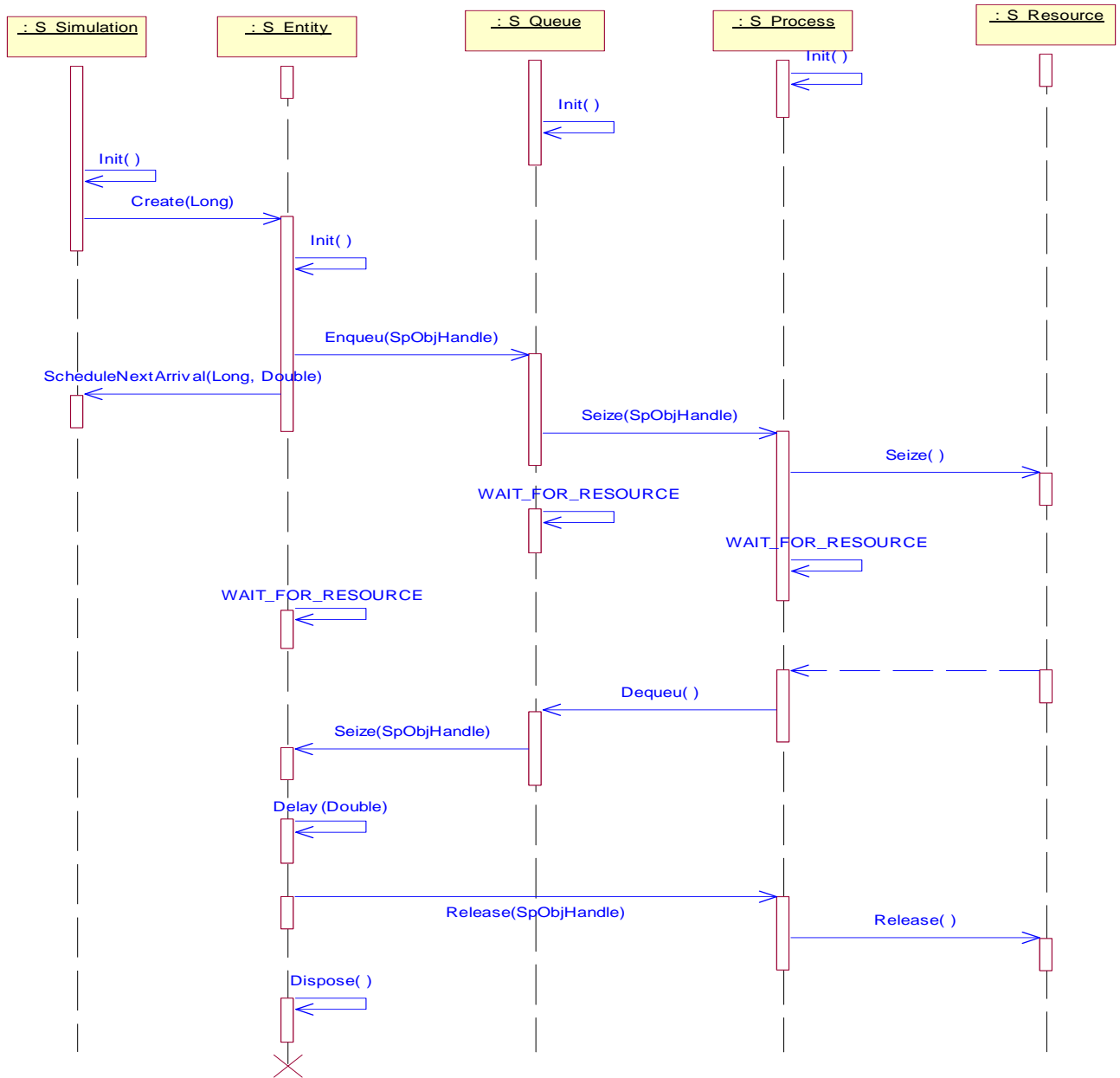


Figure 18 An Example Sequence Diagram of Simulation Classes

Visualizations in the VTB

We have identified that two types of additional visualizations are required in the context of VTB distributed simulation. First, a visualization of data and/or the specialized functions is an essential part of COTS tools, but the tools do not support any type of simulation concepts. In order to integrate the visualization tool to the VTB, we have created a federate that interacts with both the RTI and the tool's external interface which may be in such formats as Component Object Model (COM) or Dynamic Library Link (DLL). Second, a simulation engine includes a set of integrated animation facilities to display the state of the system being simulated which may allow the user to interact with the model. It does not, however, support any function for visualization of a remote federate in the federation. To address this problem we have utilized a COTS simulation package to include the state of remote federate in local federate. This section presents two approaches for integrating the visualization of a COTS GIS tool and a remote federate into VTB.

Integrating a visualization of COTS GIS tool

Toxic gas-related risk is a function of exposure duration and toxic propellant concentration or dosage that would result in casualty for all populations (including normal and sensitive people, such as asthma patients). If a disaster occurs, the toxic risk assessment will depend on factors such as the respiration of propellants, meteorological conditions, the effect of positive or negative buoyancy on the rise or descent of the released toxic propellants, the influence of atmospheric physics on the transport and diffusion of toxic propellants released in the launch area,

the population density, location, susceptibility (health categories), and sheltering for all populations within each potential toxic hazard area.

In order to effectively present (visualize) the region covered by the envelope and to accurately calculate the population at risk associated with that region we have used ArcView with Spatial Analyst as a GIS tool and have incorporated the LandScan Global Population Database which provides characteristics of the population in detail.

Geographic Information Systems (GIS) tools

ArcView provides the tools to work with maps, database tables, charts, and graphics all in one graphical user interface (GUI). ArcView is an integrated suite of advanced GIS applications. It includes: ArcMap, ArcCatalog and ArcToolbox applications that can aid in performing many GIS tasks such as mapping, data management, geographical analysis, data editing and geoprocessing. It includes a high level geographic data model for representing spatial information as features, rasters and other spatial data types. Spatial Analyst in particular, an extension to ArcView, provides a broad range of powerful spatial modeling and analysis features. Its functionality ranges from creating and querying maps, analyzing cell-based raster data, performing integrated raster vector analysis, deriving new information from existing data, querying information from the existing data and across multiple data layers, and fully integrating cell based raster data with traditional vector data.

Population Database

We used The LandScan Global Population Database, a public domain database of the World's population developed by Oak Ridge National Laboratory (ORNL) (LandScan, 2003). LandScan includes the best available census counts (usually at province level) for each country and allocates these figures into rural and urban population distributions on a 30" X 30" lat/long grid cell system. To assign values to a specific grid cell, LandScan calculates a probability coefficient for each cell, and applies the coefficients to the census counts. The probability coefficient is based on slope, proximity to roads, land cover, nighttime lights, and an urban density factor.

Integration and Visualization

By the pre-processes of the Virtual Range model, Monte Carlo model and Calpuff/Calmet/Calpost (see Chapter 4 for detail), an input to ArcView, concentration of toxicant with area under influence, is generated. The steps taken to generate the number of people exposed/casualties and a display of the area of impact follow. .

The input text file which includes area under influence as Universal Transverse Mercator (UTM) Coordinates and the ground level concentration of Hydrochloric acid (HCl) in a particular area of interest are imported on top of the population map as a data layer, and later the two layers are combined to only select the cross areas. The added point HCl layer is saved into a feature dataset for query. Then, the query is run on the saved HCl layer to select the region where the concentration of the HCl is 7 ppm, *i.e.*, 0.0104387 gm/m^3 (Sepulveda et al., 2004a). After performing the above steps, the Zonal Statistics function of Spatial Analyst is used to calculate the total number of people affected by 7 ppm of HCl. Zonal Statistics calculates the statistics for

each zone of a zone dataset based on values from another dataset. A zone is a region in which all the cells in a raster have the same value, regardless of whether or not they are contiguous. The output sum in the zonal statistics gives the total number of people affected in that 7ppm of HCl zone. Figure 19 shows the visual components of the incorporated data.

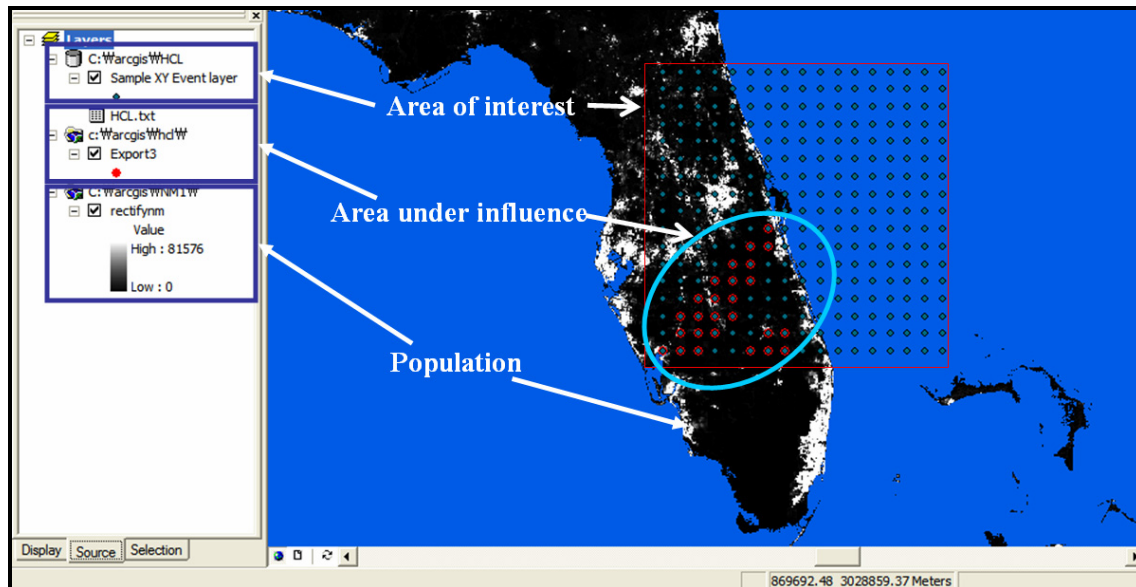


Figure 19 Calculation of Population at Risk

Animation System Architecture

The fundamental concepts and properties of visualization design are required to develop effective tools for visualization of a simulated system. A general architecture of visualization (Lin, Yeh, & Sheu, 1992) in which the common components of an animated simulation system and interactions among the components required to implement a general animation system are identified and also a Model-Animator-Scheduler paradigm is presented for unifying different approaches of animated simulation systems into a single structure and serving as a foundation for implementation and further research.

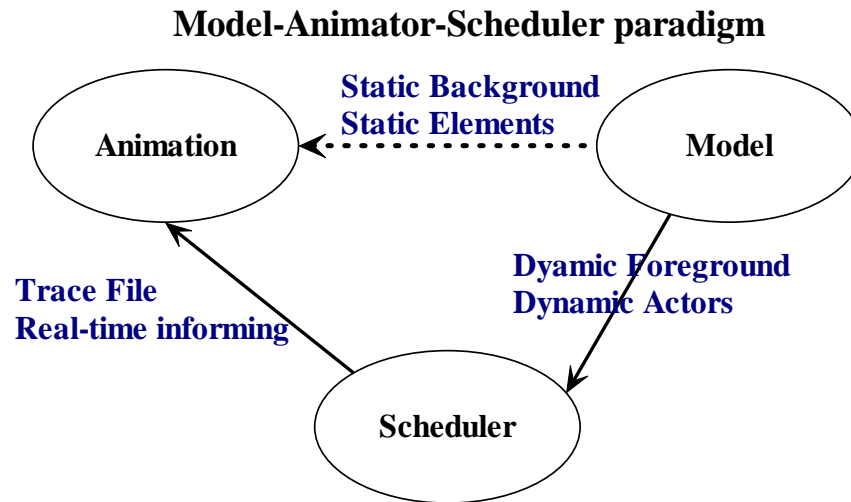


Figure 20 Model-Animator-Scheduler paradigm (Lin et al., 1992)

The major components of the paradigm are the Model, Animation, and Scheduler. The Model represents a simulation program which comprises the description of processes and entities as well as data which specify their characteristics. The Animator includes all the facilities that are required to be displayed to present the state of the system components and related tasks such as static layout of the system, moving entity, and static/dynamic resource. Scheduler coordinates the time associated event between Model and Animator. These components are linked to display the state of system simulated by the Model either as the model runs or post-simulation run. During the animation the following data are passed from the Model to the Animator. The Static Background is the representative display of the target system which never changes over the simulation run (e.g. layout). Static elements are the static objects of the target system. It can be used to reflect the difference between experiments (e.g. location of machine under the same layout). Dynamic actors are the movable objects (entities) of the system. Since the location and state of the object may change over the instance of simulation time, the update should be provided by the model. Dynamic foreground is the summary of time-varying system status or variables. Although its location is fixed, the containing values are updated by certain events. The event's trace

for each simulation run is collected and recorded to the trace file which will be used to drive animation.

A majority of effort being put forth to develop a way to visualize simulation data on the HLA-based distributed simulation has been focused on the area of interactive simulation where there generally is a common scene shared by most, if not all, of the federates. This type of visualization is good for use as a training simulation (or Man-in-the loop simulation) and a game-like simulation in which a relatively small shared space is being simulated with multiple actors joining, interacting, and resigning. A widely employed approach in visualization for HLA-based distributed simulation is a display federate paradigm in which a visualization model is implemented, and then integrated into a federation as a utility (display) federate. Most of these display federates use a dedicated visualization tool such as Proof Animation (Strassburger, 1999), Skopeo (Klein, Schulze, & Strassburger, 1998), and Quest (Roberto, Guixiu, & Charles, 2003), among others.

We have found that the visualization of a remote federate can be developed by facilitating many COTS simulation packages if the packages have embedded animation facilities and have the HLA interoperability. The visualization of a remote federate is designed with the following steps: (1) The visualization components are categorized into dynamic elements (dynamic foreground), static elements (static background, static elements), and dynamic actors which are identified in the *Model-Animator-Scheduler paradigm* (Lin et al., 1992). The classification is made based on the how the state of these elements is changed. The state (appearance) of static elements never changes during the simulation run. The state (appearance) of dynamic elements changes whenever there is an interaction with a dynamic actor. Therefore the dynamic elements

do not automatically change their state without interacting with a dynamic actor which means the elements can be pre-built into the remote visualizer.

The dynamic actor represents an active entity in the model which flows through some part of the simulated system. The entity has deterministic attributes and stochastic attributes both of which together determine the state of the entity. The deterministic attributes of an entity can be pre-defined or changed depending on the value of other attributes with it, however the conditions of the change are known at the stage of model building in both (remote federate and visualizer). Therefore the last thing we need to feed to the visualizer in order to display the state of remote simulated system is the stochastic attribute, the value of which will be drawn when the entity reaches a modeling block that acts as a *delay* or a *service*, the duration of which is stochastic. In some simulation studies, especially the study of Variance Reduction techniques, using *Common Random Numbers*, the stochastic attributes of simulation components are assigned at the beginning of the simulation run so that all the random numbers (stochastic attributes) assigned to them will be the same from one system configuration to another. If we take the same approach the implementation will be simple because the number of information transfers from the remote federate to the visualizer is the same as that of entities in the simulated system.

Despite to its simplicity, it may not be applicable if the remote federate has an external user interface e.g., a machine or human-in-the-loop simulation. The response from the external interface may vary in terms of simulation time which causes the change in the order of the random number stream and consequently the value of stochastic attributes assigned at the beginning of the simulation run may not be the same as the values assigned each time the entity reaches a modeling block.

(2) The visualizer is created in such a way that it depicts all the visual components which include static background, static elements, dynamic foreground, and the shape of the active entity as well as the deterministic model logic which may change the course of the activity and/or the appearance of all the visual components during the simulation run.

(3) The remote federate is adapted to generate and then send HLA interactions to the visualizer. An HLA interaction consists of the time of event, active entity identification, and the duration of activity. In the visualizer an active entity flows through some parts of the system until the next simulation element requires a stochastic value to process. For most cases of discrete event simulation it is simply a logical simulation time delay except the time of creation of an entity, and waiting for an HLA interaction. As soon as it gets the HLA interaction it applies the duration of the next activity and then the process. The flow is repeated until it reaches the end of the simulation run.

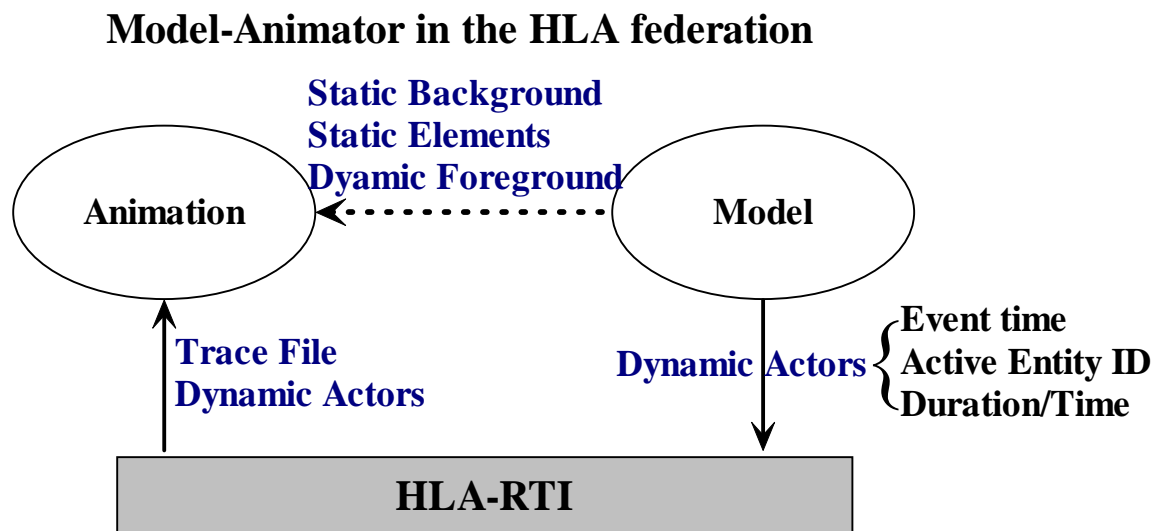


Figure 21 Model-Animation on the HLA

The correctness of visualization is achieved by the HLA's Time Management services which deliver time-stamped events to all federates in the federation. This actually simplifies the role of scheduler in *Model-Animator-Scheduler paradigm*.

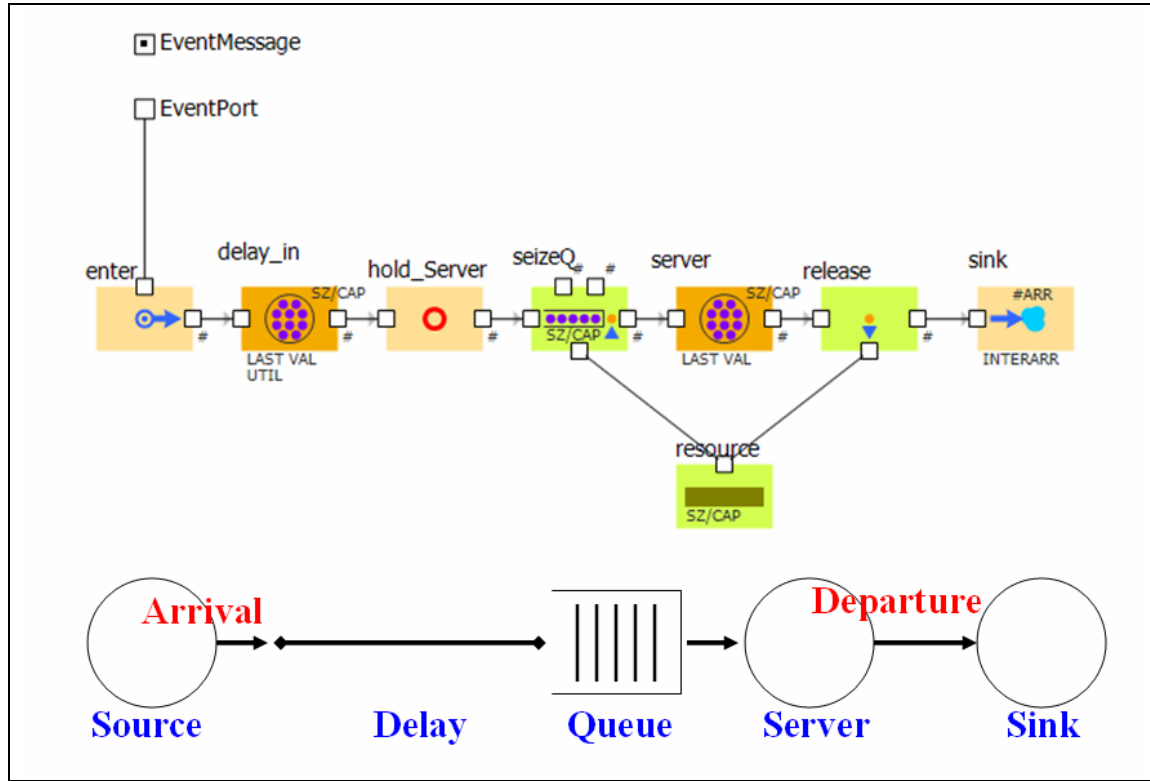


Figure 22 An Example model for a Remote Federate Animation

The main model components for an animation of remote federate consists of following “Enterprise Libraries” in AnyLogic: (1) a port with “Message type” as “HLAInteractionEvent-Message” and “Port type” as “HLAInteractionTransceiverPort”, (2) a port with “Message type” as “EntityMsg” and “Port type” as “EntityOutPortMultiple”, (3) additional “Hold” blocks as many as the number of blocks which require one or more random variates to specify the duration of delay or service, and (4) the exactly same model structure of the remote federate with generating none of the random variates which will be assigned to as the value of remote federate during the simulation run. In addition to the animation model, some adaptation code for the original re-

remote federate are required to generate and publish the value of random variables to the federate which displays the state of the original federate.

Figure 22 (top) shows an example animation model of a remote federate of a single server queueing system shown in Figure 22 (bottom). In the example, the time of entity (customer) arrival, the duration of delay, “delay_in”, and the duration of service are the places requires the random variables which should be the same as that of the original model.

The example model works as follows: (1) When each entity arrival occurs in the remote federate, it will send an interaction that contains parameters of the time of arrival and the duration of the delay, “delay_in”, then the animation federate gets the interaction through “EventMessgae” port via the RTI. The “EventMessage” port generates an entity with the parameter values, and then puts the entity to “EventPort”. The entity will pass through the “EventPort” and “Enter” block within the same logical time. When it reaches “Delay” block, the “Delay” block holds the entity as much as the parameter value of delay. (2) After finishing the delay the entity enters the “Hold” block and waits for another interaction from the “EventMessgae” port. When the “EventMessgae” port receives an interaction that contains parameters of a name of “Hold” block and a value of random variable, instead of an interaction indication “arrival of an Entity”, it creates a vector instance which contains the parameter information and then inserts the vector instance into a datastructure in the event time-stamp order. The “EventMessgae” port then sends a signal to the “Hold” block specified in the interaction to release the holding entity. The released entity will fetch the duration of next delay or service from the top vector record from the datastructure. The process of (2) will be repeated as many as the number of “Hold” blocks. The actual movement of entity, changes of dynamic background, and the status of server/resource are visualized by the facilities in the simulation package in this case AnyLogic.

Figure 23 and Figure 24 show the code samples of AnyLogic HLAInteraction class used in the visualization for Declaration Management and Object Management services.

```
import com.xj.anylogic.hlasupportmodule.*;
import com.xj.anylogic.hlasupportmodule.datacodecs.*;

class HLAInteractionClassEventMessage extends HLAInteractionClass {

public HLAInteractionClassEventMessage() {
    super( "EventMessage", null );
}
protected HLAInteractionClassEventMessage( String sClassName, HLAInteractionClass refBaseClass ) {
    super( sClassName, refBaseClass );
}
protected void OnAddParameters() {
    AddParameter( "event_type", HLADefaultDataCodecs.typeString );
    AddParameter( "object", HLADefaultDataCodecs.typeString );
    AddParameter( "time", HLADefaultDataCodecs.typeString );
    AddParameter( "time_delay", HLADefaultDataCodecs.typeString );
}
public HLAInteraction CreateInstance() {
    return new HLAInteractionEventMessage();
}
};
```

Figure 23 HLA Interaction Class for Declaration Management Service

```
import com.xj.anylogic.hlasupportmodule.*;

public class HLAInteractionEventMessage extends HLAInteraction {
    // Construction/Destruction
    public HLAInteractionEventMessage() {
        super( HLAInteractionClass.GetInteractionClass( "EventMessage" ) );
    }
    protected HLAInteractionEventMessage( HLAInteractionClass refClass ) {
        super( refClass );
    }
    // Attribute accessors : event_type
    public String getevent_type() {
        Object result = GetParameter( "event_type" );
        if ( result == null ) {
            return new String("Error_in : event_type");
        }
        return ((String)result);
    }
    ...
};
```

Figure 24 HLA Interaction Class for Object Management Service

The advantages of this approach are: (1) Reuse of the COTS simulation packages to make the animation of simulated components, which is familiar to the modeler instead using a new specialized visualization tool. (2) A visualization of remote federate can be coupled with an existing simulation model. This augmented visual information may help the distributed decision maker who is able make management decisions based on both the local and the remote information. One drawback of this implementation of animation of a remote federate may be the creation of a copy of the remote federate in the visualization model, though the complexity of the copied model is greatly depreciated. (3) It requires much less information (dynamic actor) to transmit in order to achieve the same level of resolution as the animation of remote federate. If a dedicated animation tool is used to display the scene of the remote federate, it requires transmitting not only changes on the dynamic actor but also changes on the dynamic foreground.

CHAPTER FOUR: CASE STUDIES

Factors Affecting the Expectation of Casualties in the Virtual Range Toxicity Model

The Virtual Range (VR) is “an environment that seamlessly integrates several models to simulate and visualize complex systems. In the face of a disaster regarding the Space Shuttle, there is a specific criterion that determines the launch decision. If toxic gases are released, it is necessary to predict where the gas plume will go, how far it will extend, and the expected concentration of toxins. The Virtual Range Toxicity Model’s goal is to determine the expectation of casualties (E_c) resulting from a toxic gas dispersion if a disaster occurs within the first 120 seconds of an orbiter’s liftoff” (Sepulveda et al., 2004a). This system will be able to determine the launch decision.

The area affected by the dispersion of gases is called the range. It is a system of a spaceport. According NASA-Kennedy Space Center (KSC) the range is the volume of space through which the vehicle must pass on its way to and from orbit. It is mostly used for vehicle tracking, telemetry, and communications (Barth, 2002). The range encompasses many different operations; among them, range safety has a high level of complexity. The responsibilities of the safety offices include three areas: 1. System safety reviews, 2. Flight safety, and 3. Ground safety (Committee on Space Launch Range Safety, 2000). Ground safety concerns the projection of that volume onto the surface and the people there that may be exposed in the event of a disaster. The actual dimensions of the volume and its projection onto the surface depend on the weather conditions, the vehicle’s speed and direction, and the risk component being analyzed. For example, the volume (and the corresponding projection onto the surface) for the impact of gas dispersion will

be considerably larger (and have a different shape) than the volume and projection resulting from the orbiter's resulting debris. As the vehicle moves, the range encompasses new volumes of space and leaves behind areas that fall out of range of potential hazards. The corresponding projection onto the surface also changes dynamically in size and shape.

Toxic gas-related risk is a factor of exposure duration and toxic propellant concentration or dosage that would result in casualties (death or incapacitating injury) of normal and sensitive people in a given population area. Table 2 displays the most commonly used Shuttle propellants. Public exposure to values above the ceiling concentration may cause casualties. Values in the last column reflect time-weighted average concentrations that may cause casualties.

Table 2
Commonly used Shuttle Propellants (adapted from (Sepulveda et al., 2004a))

Toxicant	Toxic Concentration	
	Ceiling [ppm]	60-min TWA [ppm]
Ammonium Perchlorate / Aluminum (solid propellant) $\text{NH}_4\text{ClO}_4 + \text{Al}$	10	2
Hydrazine	-	2
Nitric Acid (HNO_3)	4	2
Mixed Nitrogen Oxides (NO , NO_2 , N_2O_4)	4	-

The Shuttle has three fuel systems. Two are used to gain orbit and one is used to operate the orbiter. The orbiter's main engines, used to gain orbit, use hydrogen as fuel and oxygen as oxidizer, both are stored in the external tank, these components are controllable because the reaction can be stopped at any time and does not produce hazardous components. The Solid Rocket Boosters used by the second system to gain orbit use aluminum as fuel, ammonium perchlorate as oxidizer and a small amount of iron as a catalyst. This reaction produces several tons of hy-

drochloric acid (HCl). Once ignited, these components continue to burn until all of the fuel is gone. This reaction cannot be stopped and provides the lifting force for the system. At the ignition there are over a million pounds of propellants and it takes only two minutes to burn.

For the purpose of this research, the VR focuses on the health impact of the release of large amounts of HCl, a major toxicant in the event of a loss of the vehicle. The effect of exposure to HCl may range from mild irritation and headache to incapacitation due to constriction of the airway and lack of oxygen delivery to the brain. The analysis for other toxicants resulting from a Shuttle disaster will be similar.

The Virtual Range Toxicity Model's Architecture

The VR integrates a Range Safety Simulation model, Geographic Information Systems (GIS), population data, gas dispersion models, and weather information. The architecture is modular and uses Commercial Off-The-Shelf (COTS) applications such as ARENA, CALPUFF, and ArcMap so that it can be easily applied to other shuttle models and/or other launch operation areas. Figure 25 shows the architecture of the Virtual Range Toxicity Model.

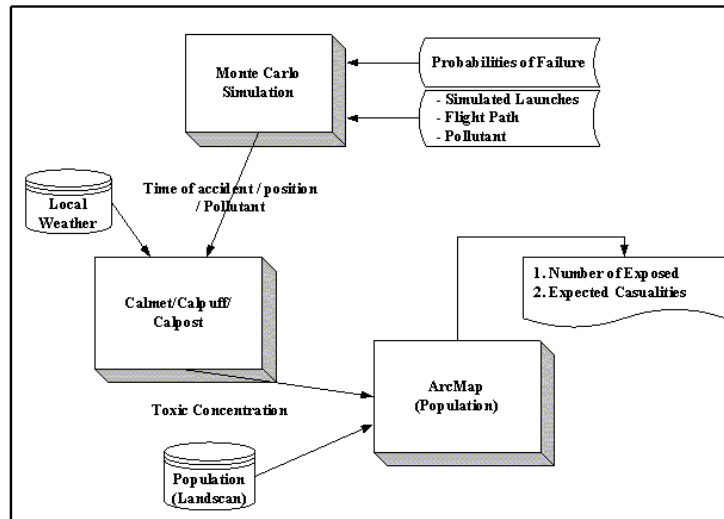


Figure 25 Virtual Range Toxicity Model Architecture (Sepulveda et al., 2004a)

The Monte Carlo simulation – a technique that repeatedly generates random values for uncertain variables to simulate a model – accounts for the effects on risk of factors such as vehicle position and consumption of propellants, weather uncertainties, vehicle guidance, and vehicle performance deviations. The need for a simulation of these factors is paramount. For example, toxic gas impact risk is affected by variability in the meteorological and launch vehicle parameters, wind uncertainties, and other weather related characteristics. The Monte Carlo simulation is also used to determine the launch decision. For any planned flight path, it is needed to determine what the Ec would be with the actual conditions (input parameters). These analyses will identify parameters with the largest impact on the value of Ec and, therefore, identify where modeling accuracy is most critical.

The VR incorporates flight trajectory data and weather information in and around KSC, a model of the toxics dispersion tailored for the NASA Shuttle at low altitudes, a GIS to visualize the area over land affected by the disaster, a population model to determine the number of people exposed in that area, and a probabilistic calculator/simulator to compute Ec.

If an accident occurs, the model determines the position, volume, and initial dispersion velocity of the released pollutants. These values are the input to CALPUFF (CALPUFF Modeling System, 2004) a multi-layer, multi-species, non-steady state Lagrangian puff dispersion model - which in turn predicts the toxic concentrations of the toxicant at a specified time after the onset of the accident. These values determine the envelope over land where the pollutant concentration exceeds the ceilings imposed by the pollutant's Exposure Response Functions (ERFs). We use the number of exposed people under that envelope to estimate the number of casualties resulting from exposure to toxic levels of the released toxic propellant for that simulated disaster.

The scope for E_c calculation is restricted to gas dispersion, for which we focus on displaying boundaries. We use as critical value the concentration defined for an $E_c = 30 \times 10^{-6}$ casualties/launch, resulting from a number of legal decisions related to carcinogens causing cancer and generally accepted for the Federal Aviation Administration. The system was also designed with a user friendly interface that provides numerical and graphical summaries of potential outcomes, with user-defined preferences for the display of units of measure, geographic locations, and time values.

Factors Affecting E_c

This section describes the model components and the static and dynamic data integrated into the VR and focuses on the factors that may significantly affect the computation of the expectation of casualties resulting from the toxic effects of a gas dispersion that occurs after a disaster

affecting a Space Shuttle within 120 seconds of liftoff. This section is adapted from (Sepulveda et al., 2004a).

Flight Path

For a planned flight path trajectory (altitude, speed, direction), the system projects an appropriate “envelope” (i.e., the footprint of the projected impact) for a given risk-component. We focus on released toxic gases and the system predicts their paths and concentration levels.

Figure 26 displays the typical launch sectors for launches from the Eastern Range (Cape Canaveral Air Force Station and KSC; owned or leased facilities on downrange sites such as Antigua and Ascension; and in the context of launch operations, the Atlantic Ocean, including all surrounding land, sea, and air space within the reach of any launch vehicle extending eastward into the Indian and Pacific Oceans.) (AST, 2002). In general, vehicles are launched in an easterly direction and on an azimuth that provides protection of land masses and populated areas on and off the facility, including the Caribbean Islands, Bermuda, the northeast coasts of South America, and Africa. For polar launches, the azimuth upper limit is 37° and the lower limit is 44° . For equatorial launches, the azimuth upper and lower limits are 110° and 114° , respectively.

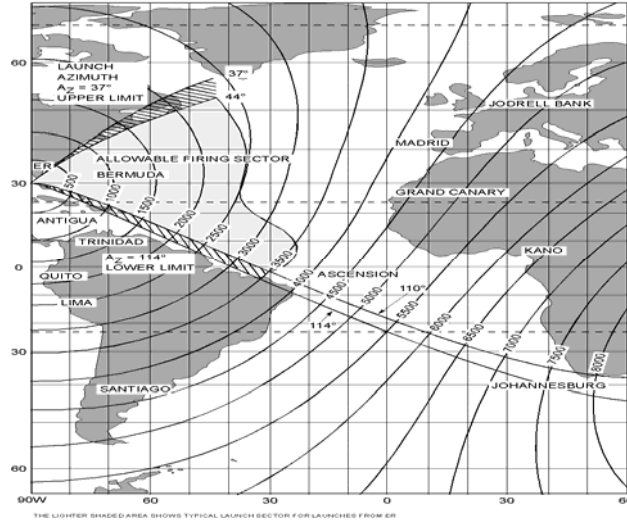


Figure 26 Launch sectors from the Eastern Range (adapted from (Sepulveda et al., 2004a)

The path of the shuttle for equatorial launches is calculated from data of past launches given in EFG (Earth-fixed geocentric) coordinates. This information was converted into latitude, longitude, and altitude assuming a spherical model of Earth. To make the conversion from EFG coordinates to longitude and latitude, we used the following formula:

$$\phi = \sin^{-1} \left(\frac{z}{\sqrt{x^2 + y^2 + z^2}} \right)$$

$$\theta = \tan^{-1} \left(\frac{y}{x} \right)$$

where ϕ is the latitude, θ is the longitude, and x, y, z are the geocentric coordinates. For the conversion to UTM (Universal Transverse Mercator) (NIMA, 2001) units we used the software Corpcon, Version 5.11.08, which allows the user to convert coordinates between Geographic, State Plane, and UTM systems on the North American Datum of 1927 (NAD 27), the North American Datum of 1983 (NAD 83), and High Accuracy Reference Networks (HARNs) (NGS, 2004).

Because of the assumption of the spherical model of the earth, the conversion gave us an error of less than 0.5% as compared to the real position of the Launch Pad (Pad39a) in KSC. The altitude was obtained using the distance from the center of the Earth to the position of the shuttle before launch as reference. Using this method the calculated altitude at which the Solid Rocket Boosters separation occurs was 44km, approximately 120 seconds after launch.

In order to meet the data requirements of CALPUFF, the resulting data was translated into UTM NAD 27 Zone 17. The UTM grid is a special grid adopted by NIMA (National Imagery and Mapping Agency) for military use throughout the world. In this grid, the world is divided into 60 north-south zones, each covering a strip 6° wide in longitude. These zones are numbered consecutively beginning with Zone 1, between 180° and 174° west longitude, and progressing eastward to Zone 60, between 174° and 180° east longitude. Thus, the conterminous 48 States are covered by 10 zones, from Zone 10 on the west coast through Zone 19 in New England (Figure 6). The first factor that may be significant is the direction (polar, equatorial) of the launch.

Probabilities of Failure for the Shuttle

The second factor is the exact location where the accident occurs. The VR interface grants the analyst the ability to select a random occurrence for the accident (e.g., to use Monte Carlo Simulation) or to “fix” the time of the accident. There is also a third time-related option, which is to specify a series of observations at fixed time intervals (for example, at 0, 10, 20, 30, etc. seconds after launch). The Monte Carlo simulation works by generating random numbers based on the probabilities of certain events occurring which are obtained from a report on poten-

tial causes for a loss of vehicle (Fragola & G.Maggio, 1995). This document presents the total probability of losing the vehicle due to the failure of the different systems and subsystems of the shuttle. In order to obtain the probability of losing the vehicle at the different stages, the first 120 seconds were divided into representative events that depict a range of time for which we calculated the probability of losing the vehicle as a result of an issue within one of the main components such as external tank, space shuttle main engine, integrated solid rocket booster, and orbiter (Table 3). With the intention of getting a better estimate of the probability at the different stages, shuttle experts were asked to assign weights to represent their best estimate for a failure occurring in a given subsystem during the shuttle operation. This questionnaire is classified according to the main subsystems: external tank, space shuttle main engine, integrated solid rocket booster and orbiter. With this information the total probability was weighted and calculated at each stage within the first 120 seconds.

Table 3

Representative Events in the Shuttle's Trajectory during the first 120 Seconds (adapted from (Sepulveda et al., 2004a))

Stage #	Start Time (sec)	End time (sec)	Interval	Event	Description
1	0	0.3	0.3	SRB Ignition	SRBs ignite. The SSMEs are at 100% rated power and gimbaled to launch position; all connections with the vehicle retract or are dropped
2	0.3			Liftoff	The vehicle lifts off the pad
3	0.3	6	5.7	Tower clear	Make the necessary corrections to remain in vertical flight
4*	6	10	4		
5	10	18	8	Start roll maneuver	The Shuttle begins a roll program to achieve a northeasterly track from KSC, heading toward a 51.6 degree inclination to the equator.
	18			End roll	The shuttle completes the programmed roll maneuver and is now positioned heads down, wings level.
6*	18	26	8		
7	26	30	4	Start throttle down	The three liquid-fueled main engines are throttled down to 72 percent rated thrust to ease the vehicle's flight through the dense lower atmosphere.
	30			Throttle down complete	
8*	30	60	30		
	60			Max Q	The maximum dynamic pressure reaches 580 psf
9	60	64	4	Throttle up	Main engines begin throttling back up. The engine's thrust level will be taken to 104.5 percent.
10*	64	120	56		
11**	120	126	6	SRB staging	Having consumed all their propellant, the solid rocket boosters are jettisoned from the attachment points on the external tank. The boosters parachute into the Atlantic Ocean for recovery and reuse.

The Toxicity Model

Shuttle Toxicants: In order to launch the shuttle into space, the shuttle relies upon two Solid Rocket Boosters (SRBs). The SRBs contain aluminum powder as fuel and ammonium perchlorate as its oxidizer. Hydrochloric acid (HCl) is a major combustion product.

Due to its relative quantity, the expected dispersion of HCl gas (density 1.26) is the major

determinant of shuttle launch decisions. The gas is initially exhausted as an aerosol, which dissipates within a few minutes of flight and remains as gas. During normal operation of the shuttle total exhaust of HCl is 163.3 tons during the first 15 kilometers of flight. About 72.5 more tons are exhausted by two minutes after launch (AIAA, 1991). In the event of a disaster, the SRBs separate from the shuttle, burning like “Roman candles” as they fall. If a disaster occurs close enough to lift-off, it is possible under some meteorological conditions that the ground concentration would exceed the 10 parts per million, which is the limit set for a ten-minute exposure. The short-term exposure limit (STEL) for HCl is 7 ppm (Hill Brothers Chemical Company, 2001). This irritating exposure can result in constriction of the upper respiratory tract..

The Gas Dispersion Model: The health impact of the release of large amounts of hydrochloric acid, a major toxicant in the event of a loss of vehicle, may be catastrophic. The effect of exposure to HCl may range from mild irritation and headache to incapacitation due to constriction of the airway and lack of oxygen delivery to the brain. If a “loss of vehicle” event occurs close enough to lift-off, it is possible under some meteorological conditions that the ground concentration would exceed 7 ppm, the short-term exposure limit (STEL) for HCl for normal people (Hill Brothers Chemical Company, 2001). For HCl, mild symptoms include irritation and headache, which are reversible within 48 hours and do not interfere with normal activity or require medical attention (Philipson, 1999). Moderate symptoms include cough and shortness of breath, and medical attention might be necessary. Severe symptoms include disorientation due to constriction of the airway and consequent shortfall in delivery of oxygen to the brain; changes to lung tissue are irreversible in this category. Of course, the STEL values for sensitive people (children, the elderly, and people with asthma or other respiratory diseases) are even smaller.

For the evaluation of the gas dispersion and toxic effect we use CALPUFF, developed and distributed by Earth Tech, Inc (Earth Tech, 2000). CALPUFF simulates the effects of time and space by varying meteorological conditions on pollutant transport, transformation, and removal under inhomogeneous and non-stationary conditions. CALPUFF has modules to assess toxic effects of specific chemical agents and factors such as variability of meteorological conditions, dry deposition and dispersion over a variety of spatially varying land surfaces, low wind speed dispersion, or wet removal of the pollutant.

There are several factors associated with CALPUFF that may affect the value of E_c , the most important of which are the initial speed of the toxic plume, the weather conditions (humidity, temperature, etc.), the wind speed and the direction.

The Weather Factor: CALPUFF modeling system was developed as part of a study to design and develop a generalized non-steady-state air quality modeling system for regulatory use. CALPUFF (a puff model) has recently been accepted by the US EPA as a guideline model to be used in all regulatory applications involving the long-range (>50km) transport of pollutants. It can also be used on a case-by-case basis in situations involving complex flow and non-steady-state cases from fence-line impacts to 50 km (NIWAR, 2004). It includes three main components: CALMET, CALPUFF, and CALPOST. In addition, it also includes several preprocessing programs to interface the model to standard, routinely-available, meteorological, and geophysical datasets.

CALMET is a meteorological model that develops hourly wind and temperature fields on a three-dimensional grid modeling domain with associated two-dimensional fields such as mixing height, surface characteristics, and dispersion properties. CALPUFF is a multi-layer, multi-species non-steady-state puff dispersion model which can simulate the effects of time and space

by varying meteorological conditions on pollutant transport, transformation, and removal. This is done by means of the fields generated by CALMET; or as an option, it may use simple meteorological data without the grid. Selected temporal and spatial variations in the meteorological fields are explicitly incorporated in the resulting distribution of puffs throughout a simulation period. CALPUFF contains algorithms for near-source effects such as building downwash, transitional plume rise, partial plume penetration, and sub-grid scale terrain interactions, as well as longer range effects such as pollutant removal (wet scavenging and dry deposition), chemical transformation, vertical wind shear, over-water transport, and coastal interaction effects. It can accommodate arbitrarily-varying point source and grid area source emissions. Most of the algorithms contain options to treat the physical processes at different levels of detail depending on the model application. The primary output files from CALPUFF contain either hourly concentrations or hourly deposition fluxes evaluated at selected receptor locations.

CALPOST is used to process the output files produced by CALPUFF and to summarize the results of the simulation. When performing visibility-related modeling CALPOST uses concentrations from CALPUFF to compute extinction coefficients and related measures of visibility, reporting these for selected average times and locations.

Puff models represent a continuous plume as a number of discrete packets of pollutant material. Most puff models evaluate the contribution of a puff to the concentration at a receptor by sampling at particular time intervals (sampling steps). The total concentration at a receptor is the sum of the contributions of all nearby puffs averaged for all sampling steps within the basic time step. Depending on the model and the application, the sampling step and the time step may both be one hour, indicating only one “snapshot” of the puff is taken each hour. A traditional

drawback of the puff approach has been the need for the release of many puffs to adequately represent a continuous plume close to a source.

CALPUFF uses either of the following two alternatives to the conventional sampling function: a sampling scheme that employs radially symmetric Gaussian puffs and a scheme that uses a non-circular puff (a “slug”), elongated in the direction of the wind during release to eliminate the need for frequent releases of puffs (Earth Tech, 2000).

CALMET requires four types of input files: Surface Meteorological Data, Upper Air Data, Overwater Observations, and Geophysical Data (Earth Tech, 2000). The weather information gathered corresponds to days in 2002 in which a launch took place, specifically: March 1, April 8, June 5, October 7, and November 23.

In simulating future launches for a given launch window (projected day and time for launch), we will gather similar weather information that occurred for the same time frame within the same week in the previous three years and use the average and extreme values observed for the simulation.

Surface Meteorological Data. The surface meteorological observations were obtained from (NOAA, 2004). These meteorological data files contain hourly observations of: Wind speed, Wind direction, Temperature (part of surface data file), Cloud cover, Ceiling height, Surface pressure, Relative humidity, and Precipitation type code. The Surface Meteorological Data requires information from different nearby stations for more accuracy given the position and code of the station. We used the data from four different stations in Florida. The WBAN codes and locations are as follows: MCO in Orlando, DAB in Daytona Beach, ORL in Orlando, and MLB in Melbourne. None of these stations had the surface pressure, cloud cover or precipitation type

code so it was necessary to use the default values. CALMET requires hourly information for all of these fields (NOAA, 2004).

Upper Air Data. This set of observations contains twice-daily observed vertical profiles of: Wind speed, Wind direction, Temperature, Pressure, Elevation. The data was obtained from station XMR in Cape Kennedy from NOAA's Radiosonde Database Access (NOAA, 2004).

Geophysical Data. The data file contains the geophysical data inputs required by the CALMET model. These inputs include: Grid fields of terrain elevations, Land use categories, Surface roughness length, Albedo, Bowen ratio, Soil heat flux constant, Anthropogenic heat flux, and Vegetative leaf area index.

Over-water Data. This data is necessary to know the Overwater transport and dispersion. For this purpose it is necessary to have the following information: Air-sea temperature difference, Air temperature, Relative humidity, Overwater mixing high, and Wind speed and direction.

The location of the overwater site is specified for each observation. The information collected was taken from the closest buoy, in this case Station 41009 - CANAVERAL 20 NM East of Cape Canaveral. This information has been obtained from the National Data Buoy Center, a division of the NOAA. (NOAA, 2004) (see also (Gesser R, 2003) for information about the use of overwater observations in CALMET).

Some of the data that was found in NOAA did not match the requirements of CALMET, so we calculated the missing parameters with theoretical formulas. For instance, the Relative humidity was not part of the information found in the NOAA's files. Therefore, we used the vapor pressure, the saturation vapor pressure, the dewpoint temperature, and the ambient temperature found from NOAA to calculate it.

In the VR interface, the analyst is given the option of selecting any of the given dates. The analyst can also change the default values for the wind speed and direction for any selected day. As an alternative, the “All days” option may be selected which results in one independent simulation run for each of the selected weather profiles.

Geographic Data and Population Models

The VR uses ArcGIS – a powerful commercial GIS application that provides data visualization, query, analysis, and integration capabilities along with the ability to create and edit geographic data – to identify the region covered by the dispersed gas.

The area covered in our simulation is basically the area around the Cape Canaveral region, which includes mainly Brevard and Orange Counties and a large part of the sea around the Cape. The simulation covers about 150 km in each direction from the source (Cape Canaveral). Since this area is a flat, noncomplex terrain and surrounded by sea, it has a good flow of winds, pressure, and temperature variations through it. So, the weather data plugged into the model plays an important role in the simulation. The area covered by the simulation is divided into a number of grids with equal spacing to facilitate the study of concentrations of the explosions in the area considered. Each grid can be a square block, whose side can range from 10s of meters to 100s of kilometers.

Population Model. Using the LandScan Global Population Database, a public domain database of the world’s population developed by Oak Ridge National Laboratory (ORNL), to present population data associated with the covered region, the VR determines the population at risk for that specific risk-component (LandScan, 2003). LandScan includes the best available

census counts (usually at province level) for each country and allocates these figures into rural and urban population distributions on a 30" X 30" lat/long grid cell system. To assign values to a specific grid cell, LandScan calculates a probability coefficient for each cell and applies the coefficients to the census counts. The probability coefficient is based on slope, proximity to roads, land cover, nighttime lights, and an urban density factor.

Exposure Response Functions: Figure 27 shows Exposure Response Functions (ERFs) for HCl for sensitive and normal people subject to a 30-minute exposure. The sensitive population was defined as children through age 14 and adults aged 75 and over, as well as all others with respiratory illnesses. In Brevard County, recent census data shows that 42% of the population is composed of those either those 18 and younger or those 65 and older; this number is expected to increase by about 55% by the year 2010 (United Way of Brevard County, 2002). These curves show that concentrations of 15 ppm and 41.5 ppm of HCl result in an expectation of casualties of about 30 in a million launches ($E_c = 30 \times 10^{-6}$) for sensitive and normal people, respectively. ERF curves have been computed for nitric acid aerosol, nitrogen dioxide, and hydrochloric acid (Philipson, 1999). They were constructed by a panel of about 20 expert toxicologists who provided best estimates of the 1- and 99-percentiles of expected casualties. Below the first percentile, “essentially no one in a population of a given sensitivity category would be affected to a given level of severity.” Above the 99th percentile, “essentially all in the population would be so affected.” Twelve estimates (with ranges of uncertainty) for each substance and duration of exposure (10, 30, 60, and 120 minutes) were provided by members of the panel of experts: one for each percentile, casualty type (mild, moderate, and severe), and victim type (sensitive, normal). Some of the panelists computed duration estimates from 1-hour estimates according to Haber’s Law, which states that “an effect level is directly proportional to exposure

concentration multiplied by time” (Philipson, 1999). Once these estimates were decided upon by the panel, ERF curves were then calculated as cumulative distributions.

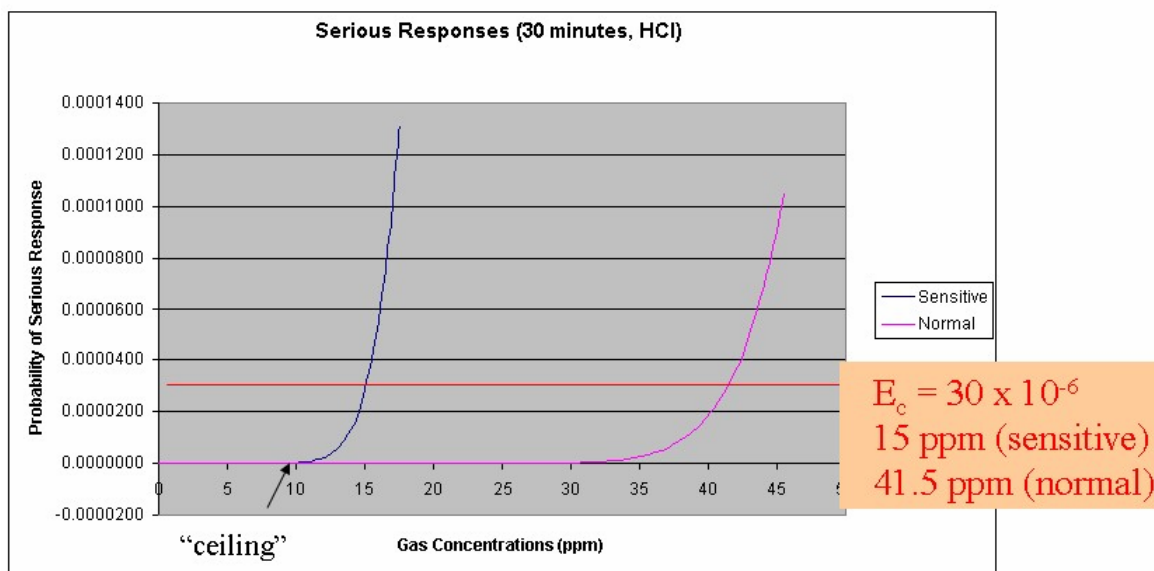


Figure 27 Exposure Response Function for HCl (adapted from (Sepulveda et al., 2004a)

Geographic Data Model. ArcGIS is used along with LandScan Global Population Database. In this GIS environment, the model of population distribution is integrated with the gas dispersion model to calculate E_c for that risk component given a loss of vehicle. Spatial Analyst, an extension toolset in ArcGIS, is used to generate the query on the HCl data from the Gas Dispersion Model to select the region where the concentration of the HCl exceeds a critical value. Zonal Statistics calculates the statistics for each zone of a zone dataset based on values from another dataset. A zone is a region in which all the cells in a raster have the same value, regardless of whether or not they are contiguous. The sum of the output gives the total number of people affected in that critical HCl concentration zone.

For the VR, the sensitive and normal HCl severe ERFs were combined according to the sensitive and normal population mix in Brevard County, Florida. A critical value of 15 ppm was

used as a baseline. This value represents a value where most sensitive people will be affected but most normal people will not. In the sensitivity analysis we will vary this factor by increasing the critical HCl concentration by increments of 10 until we reach 45 ppm, a value where almost the whole population will be affected (Sepulveda et al., 2004a).

Note that Spatial Analyst and LandScan combine to give an estimate of the number of people that may be exposed in the affected area. However, this figure represents an upper limit for the number of people at risk as some people will undoubtedly be able to take cover or flee the region before the gas dispersion reaches it. Still a sensitivity analysis could be done on the proportion of the exposed people that will actually die or be incapacitated as a result of the accident.

Integration of the Components Models

The Virtual Range Model in summary works as follows: An Arena model simulates the time of accident, which is determined by the cumulative probability of an accident occurring in ten different stages during a launch. Each of these stages has a different duration and chance of accident. Once the stage is determined, the time of accident is fixed by equal chance within the stage. Based upon the time of accident, the model references coordinates for the path of orbiter and determines the volume of remaining pollutants from the existing model data file. These values are the input to CALPUFF, which in turn predicts the toxic concentrations for each toxicant after one hour. We enter these values as a layer into ArcMap, to determine the envelope over land where the pollutant concentration exceeds the ceilings imposed by the corresponding ERF. ArcMap's Spatial Analyst has the ability to determine the number of people covered by the displayed layer. We use the number of exposed people and the parameters resulting from the pollut-

ant's ERF (we use as critical values the concentrations defined by $E_c = 30 \times 10^{-6}$) to estimate the number of casualties for that simulated disaster resulting from exposure to toxic levels of the released toxic propellant. Figure 28 shows the VR simulation model user interface. Repeating the procedure for enough simulation runs, we can get enough information to generate an “average” boundary and its associated confidence interval.

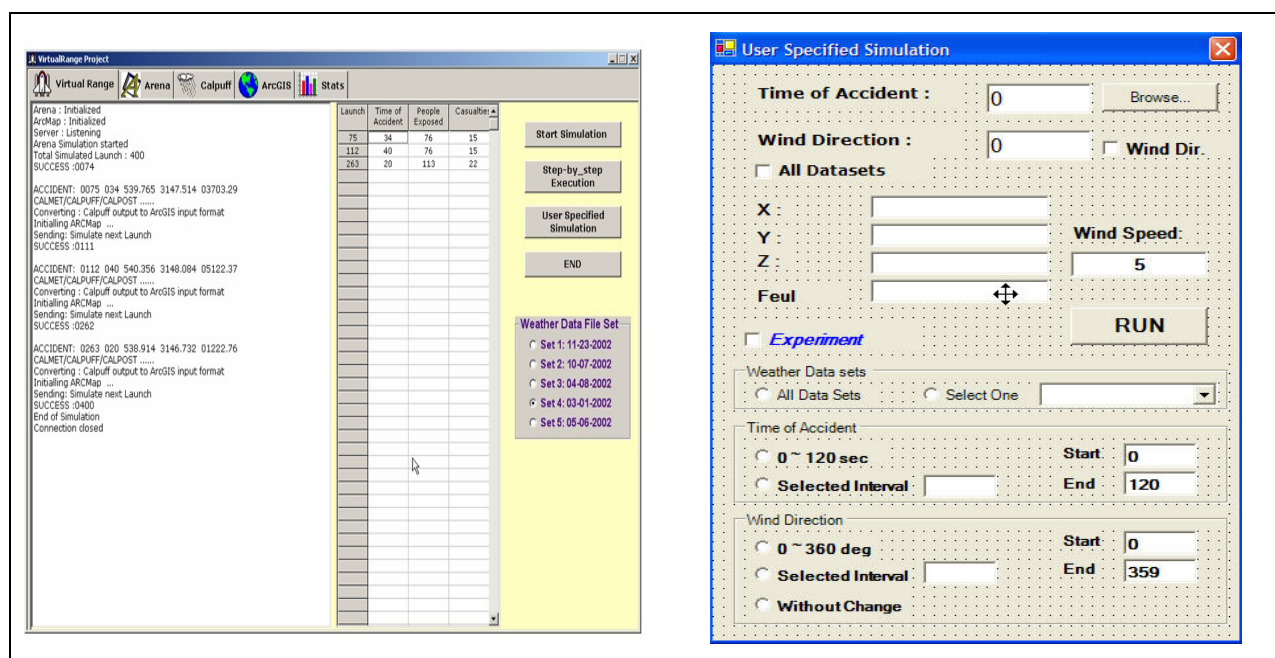


Figure 28 VR Simulation Interface

In our initial results, the sensitivity analysis shows that wind direction and the time of accident (seconds after launch) have the most significant impact on the number of people on the ground exposed to dangerous concentrations of the toxicant one hour after the onset of the disaster. In these runs, however, we varied wind direction from 0° to 360° in increments of 45° . In the final runs the limits for variation in wind direction will be given by the extreme values observed in actual launch dates.

The factors selected for the final analysis are summarized in Table 4. The dependent variable in the analysis is the expected number of people on the ground exposed to dangerous concentrations of the toxicant one hour after the onset of the disaster.

Table 4
Factors affecting Ec (Sepulveda et al., 2004a)

Factor	Example	Range
Flight path's azimuth (direction of the launch)	112°	Polar: 37°-44°; ecuatorial: 110° -114°
Time of accident [seconds after launch] (sets altitude and amount of pollutants released)	15	0-120; will try 0, 5, 10, 15, 20, 25, and 30 seconds.
Nature and amount of the released toxicant (depends on initial value, flight time, and consumption rate)	HCl, 8 tons	See Table 2.
Initial velocity of the gas plume	380 (CALPU FF's default)	Needs further research
Weather conditions	4/8/2002	As represented by 5 actual launch dates
Wind direction (f(altitude))	200 degrees	Limits represented by angles observed in actual launch dates
Wind velocity (f(altitude))	10 m/s	Limits represented by speeds observed in actual launch
Critical concentration for the pollutant	15 ppm	15 – 45 ppm (from ERF for HCl)
The proportion of exposed population incapacitated or dead as a result of the accident	60%	0 to 100%

This case study presented the factors we have selected for an in-depth sensitivity analysis of the population at risk, including vehicle trajectory, accident location, vehicle position and consumption of propellants, weather and wind uncertainties, and amount and type of toxicants released. Such factors may significantly affect the computation of the population exposed and the corresponding expectation of casualties resulting from the toxic effects of the gas dispersion that occurs after a disaster affecting a Space Shuttle within 120 seconds of liftoff.

Implementing the High Level Architecture in the Virtual Test Bed

The distributed simulation implementation described in this section represents different systems that interact in the simulation of a Space Shuttle liftoff. This example implementation displays the collaboration of a simplified version of the Space Shuttle Simulation Model and a simulation of the Launch Scrub Evaluation Model (Sepulveda et al., 2004b). The implementation follows the HLA as the principal framework to integrate all the different types of models that need to be a part of the VTB. For example, a spaceport can be represented using different types of models using different information spaceport size and operation. The simulation system will be a subsystem that will evolve over time to meet this important requirement.

Many factors contribute to a launch vehicle launching on time. The launch vehicle, spacecraft, and supporting range must all be ready to go on time in order for the launch to occur. Each of these elements may have supporting systems consisting of hundreds of subsystems and millions of individual components. Thousands of opportunities exist for technical system failure or human error. Other factors such as weather and launch area intrusions are out of the control of the launch officials. The different elements affecting launch decisions are addressed through two simulation models that were built independently.

The first model simulates Space Shuttle flow from processing at the Orbiter Processing Facility (OPF), through transport to the launch pad, liftoff, mission, landing at KSC, and refurbishing at OPF to get ready for a new launch. This is a simplified version of the conceptual flow diagram described by the Space Shuttle Processing Model (Cates, Steele, Mollaghasemi, & Rabadi, 2002). A single shuttle is used to route between the different facilities and launch opera-

tions at KSC. All processing times come from real world data as included in the Space Shuttle Processing Model.

In the Space Shuttle Processing Model, when the orbiter reaches the launch pad and is ready for launch, the simulation generates a random variable to determine the time that will elapse until the launch occurs. This time follows a theoretical distribution that closely matches events as historically observed. Those events account for historically observed instances of delays or scrubs that affected the launch process. A delay means the launch is postponed for a short time but still occurs on the expected date. A scrub means the launch is postponed for at least one day. To illustrate the VTB capabilities and the procedure needed to combine existing computer simulations, the randomly generated delay (or scrub) in the Space Shuttle Processing Model was deleted, and processing requiring Shuttles to wait on the launch pad until an external authorization for launch is received was added. To generate launch authorization commands, a second model independently simulates the range, the launch pad, and other spaceport facilities. This model focuses on events occurring in the range and in the processing facilities that can cause launch delays or launch scrubs due to mechanical or electrical failure. Although both models discussed here were built using Arena, either one of them (or both) could have been built using other software such as ProModel™, Anylogic™, or any other commercial simulation software that supports an interface to the RTI.

The Space Shuttle Simulation Model

The Space Shuttle Simulation Model (“LaunchPad”, here after) is a mini model of space shuttle operations in Arena. Here we use a single shuttle to route in between different facilities

and launch operations at KSC. The Shuttle starts the processing from the OPF (Orbiter Processing Facility). All processing time comes from real world data. After OPF processing, the shuttle is routed to the PAD where it completes the PAD processing and sends the signal to the Launch Delay and Scrub Model (“MissionControl federate”, here after) that it is ready to launch. At this point in time, the LaunchPad will wait for a GO/NOGO signal from the MissionControl federate. This signal passing takes place through RTI. As soon as the LaunchPad gets the signal from the MissionControl federates to launch, it will route the shuttle to orbit, where it will finish the orbiting process. At the end of the orbiting process, the model checks for the end-of-mission day and lands the shuttle at KSC. After the shuttle lands at KSC the model checks the shuttle’s flight number. If the flight number is 8, the shuttle is sent to Palmdale for maintenance. Otherwise it will continue the cycle from OPF. If it is sent to Palmdale, it finishes the Palmdale processing and returns back to OPF.

The Launch Delay and Scrub Model

The Launch Delay and Scrub Model (“Mission Control federate”) represents the scrub and delay logic with their probabilities in shuttle launch. The probability for scrub and delay come from real world data. The MissionControl federate handles the logic and data.

We have some historical averages of system failures per month. A system failure was identified as a system or component failure that would result in a launch scrub. Launches can continue with many individual components or subsystems not operating as long as they have a backup or are not mission critical or safety critical mandatory items. Many factors (see Table 5 for detail) contribute to a launch vehicle launching on time. The launch vehicle, spacecraft, and

supporting range must all be ready to go on time in order for the launch to occur. Each of these elements has supporting systems consisting of hundreds of subsystems and millions of individual components. Thousands of opportunities exist for technical system failure or human error.

Table 5

Factors affecting delays and scrubs (adapted from (Lebo & Woltman, 2002))

System	Subsystem	Failure Rate
Launch Vehicle	Airborne Systems	1 failure per month
	Ground Systems	3 failures per month
Spacecraft	Airborne Systems	0.5 failures per month
	Ground Systems	2 failures per month
Range	Telemetry Systems	1 failure per month
	Tracking Systems	2 failures per month
	Command Systems	1 failure per month
Other factors	Weather	Lookup table – varies by month
	Launch Area clear	Lookup table – varies by month

A system failure was identified as a system or component failure that would result in a launch scrub. Launches can continue with many individual components or subsystems not operating as long as they have a backup or are not mission critical or safety critical mandatory items. For the launch vehicle, there was a 10.5% chance of the launch vehicle element causing a scrub. For the spacecraft, there was a 6.8% chance of causing a launch scrub. Other factors such as weather (see Figure 29) and launch area intrusions (for example, a pleasure boat or an unauthorized aircraft entering a restricted area, see Figure 30) are out of the control of the launch officials.

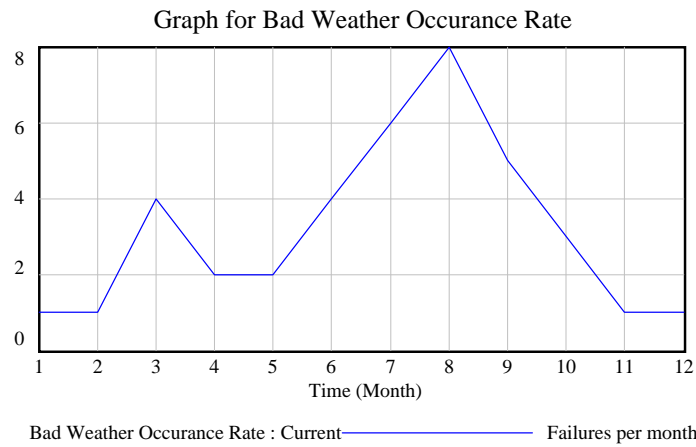


Figure 29 Bad Weather Occurrence (adapted from (Lebo et al., 2002))

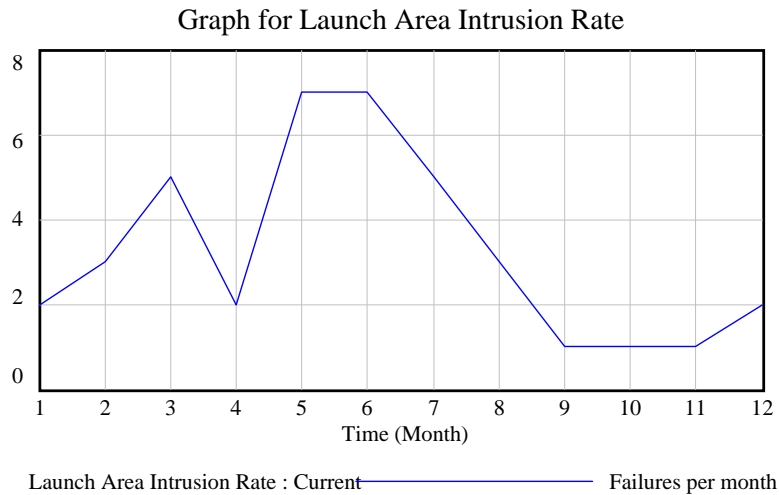


Figure 30 Launch area intrusions (adapted from (Lebo et al., 2002))

All of the hardware systems had a constant failure rate, but two items, weather and launch area clearance varied significantly with the time of year. In these cases, lookup tables were created to model the average “bad occurrences” per month for each month of the year. A simplified model (see Figure 31 depicting the different contributions and how they should be added) was built using a System Dynamics approach.

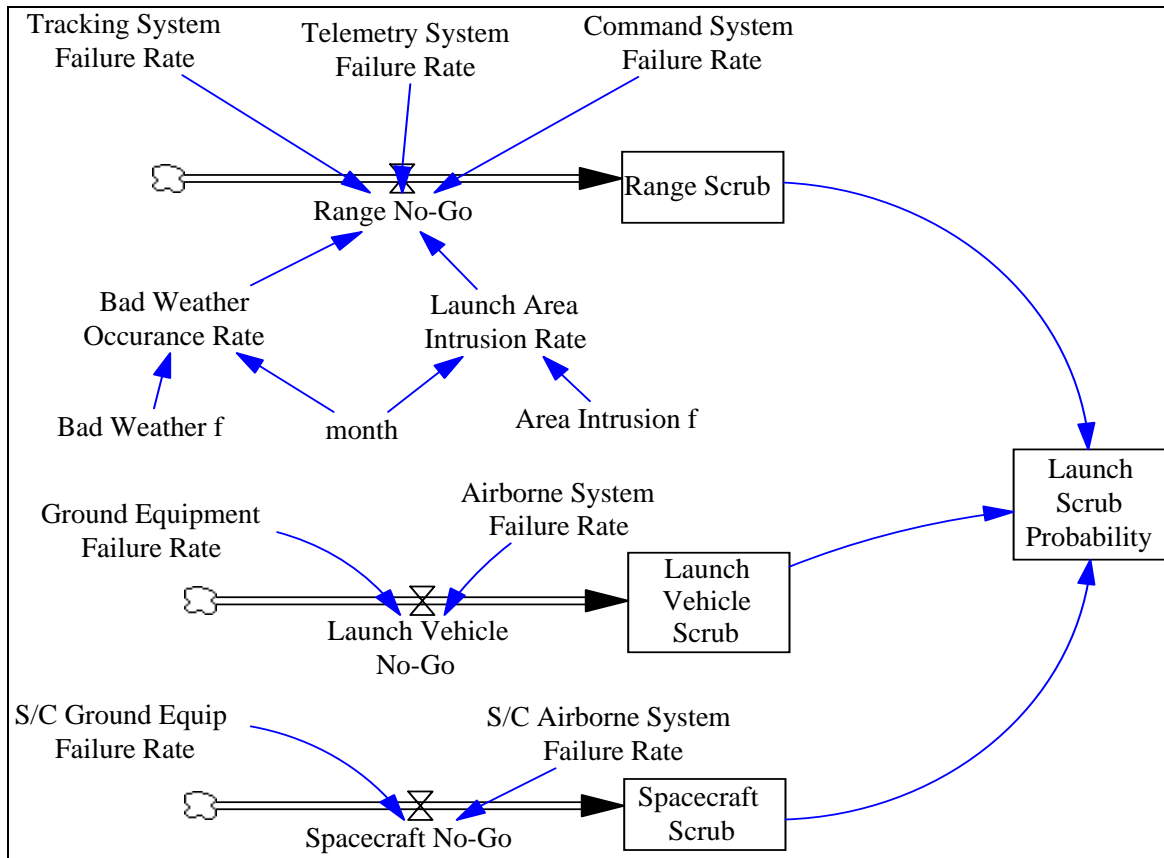


Figure 31 Contributions to the delays and scrubs (adapted from (Lebo et al., 2002))

For the range, the combined contribution of weather and range intrusions is varied by month since the weather and launch area surveillance components also varied. The probability varied from 10% to 30% depending on the month. The spring and summer months showed a higher chance of launch scrubs. The overall launch scrub probability is shown in Figure 32 and it varies between 16 and 32% depending on the month. This data could be helpful for financial and schedule planning for launch vehicles.

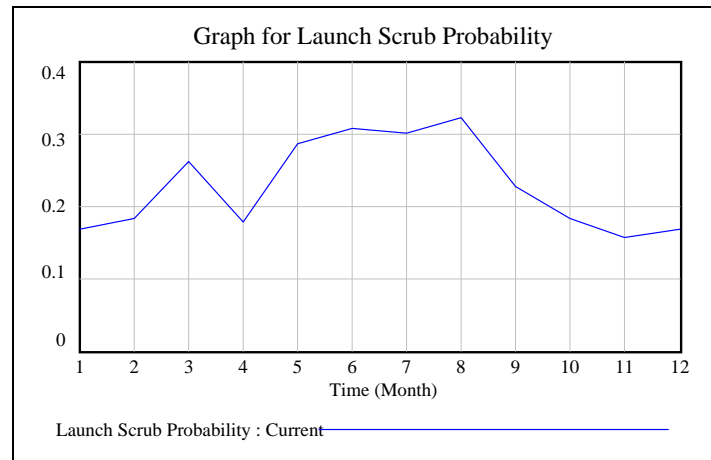


Figure 32 Overall Probability of a delay or scrub (adapted from (Lebo et al., 2002))

Integration of Space Shuttle Modell and Launch Delay and Scrub Model

The integration that occurs between the modified Space Shuttle Simulation Model (LaunchPad federate) and the Launch Delay and Scrub Model (MissionControl federate) is accomplished using DMS Adapter. The DMS Adapter's infrastructure was designed to support the integration of different manufacturing simulations with each other and with other manufacturing software applications. Applications that might be integrated using the DMS Adapter include: new or existing simulation created with existing, non-HLA-compliant simulation development tools; existing enterprise software applications dealing with non-simulation situations (production planning, human resources, inventory control, supply chain information, finance and accounting, instruments data collection, etc); or general non-simulation and non-manufacturing oriented legacy software applications. If incorporated into each federate, the DMS Adapter works with the RTI to manage the exchange of object and interaction information between federates.

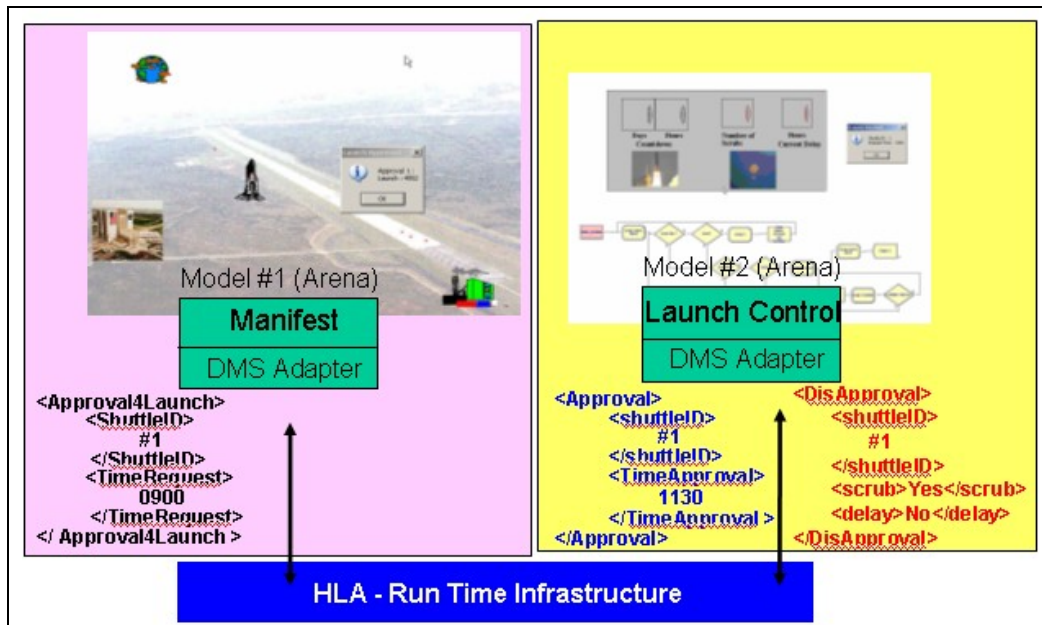


Figure 33 Distributed Shuttle Process Simulation using the DMS Adapter

This case study demonstrated the integration of models built in COTS simulation package, in this case Arena, using DMS Adapter as a gateway between the model and the RTI on the HLA distributed simulation environment. This integration can be extended to develop a new and unique collaborative computing environment where simulation models can be hosted and integrated in a seamless fashion. The focus of our future work will be to integrate and develop the Virtual Test Bed with the Virtual Range. The emphasis is on the integration of the VTB operations and the Virtual Range models. The modular architecture of the VTB which can be integrated enables the analysis of new vehicle types (e.g., the Crew Exploration Vehicle (CEV)) and the study of other launch sites. It is anticipated that the current environment will be extended to support the integration of other discrete-event simulations of KSC operations, and to make greater use of the High Level Architecture.

CHAPTER FIVE: A PROTOTYPE IMPLEMENTATION OF VTB SIMULATION SYSTEM

Introduction and Motivation

This chapter first presents a successful implementation of a prototype VTB simulation system for a proof of concept using a distributed simulation engine - the Run Time Infrastructure (RTI) of the High Level Architecture (HLA); the HLA interoperability supporting tools include the SPEEDES gateway, Distributed Manufacturing Simulation (DMS) Adapter, HLA Support Module (HSM); and simulation languages such as SPEEDES, Arena, and AnyLogic.

The complex nature of VTB is a result of the combination of multiple simulations and non-simulations (supporting tools), and a mixture of analytical discrete event simulation models which generally can be executed “as-fast-as” possible and real-time factors as an input for the model are only available through live applications (e.g. Weather Expert System). These factors make it difficult to integrate into the VTB simulation architecture. The VTB federation consists of the shuttle process federate, weather expert federate, Mission Control federate with the augmented visualization of the federation, Monte Carlo federate, and virtual range federate. The preliminary VTB simulation system manifests the requirement of VTB with respect to the distributed simulation interface on the HLA-RTI. By incorporating these federates, each of which represent operations in VTB modeled in various tools and computing environments, the VTB architecture can be used to analyze more complex, larger operations and provide associated solutions such as structural process or cost optimization. This integration is accomplished by providing the HLA interface at the communication level as well as the data representation level for multiple

simulation models. The purpose of prototype implementation is not only to demonstrate the proof of concept, but also to validate the design approach for developing the VTB simulation environment.

Adapting Legacy Models to VTB Simulation System Using the HLA

Virtual Range Toxicity Model

As described in Chapter 4, the Virtual Range toxicity model is an integrated set of software packages that exchange information in order to calculate the Expectation of Casualty as a result of gas dispersion when an accident ending in loss of vehicle affects the Space Shuttle within 120 seconds of liftoff. Among these software packages is a MonteCarlo simulation, a gas dispersion model (Calpuff), a population model (LandScan), a Geographical Information System (ArcView and ArcGIS Spatial Analyst) and access to weather data and flight path information.

The VR toxicity model was divided into two simulation models (federates), MonteCarlo federate and VR federate. The MonteCarlo federate simulates the time of accident, which is determined by the cumulative probability of an accident occurring in ten different stages during a launch. Each of these stages has a different duration and chance of accident. Once the stage is determined, the time of accident is fixed by equal chance within the stage. Based upon the time of accident, the MonteCarlo federate references coordinates for the path of orbiter and determines the volume of remaining pollutants from the existing model data file.

Each request of simulated launch from other federates the MonteCarlo federate determines whether the launch is successful or whether a simulated disaster occurs. If there is a suc-

successful launch the Monte Carlo federate sends an interaction indicating “successful launch” through the RTI; if there is an accident it sends an interaction which include the location (latitude, longitude, and altitude) and the concentration of toxicant released to the federation. All other components of the Virtual Range toxicity model are included in the VR federate and they work the same as before. Figure 34 shows the new architectures of the Monte Carlo federate and the VR federate which are adapted to the HLA distributed simulation.

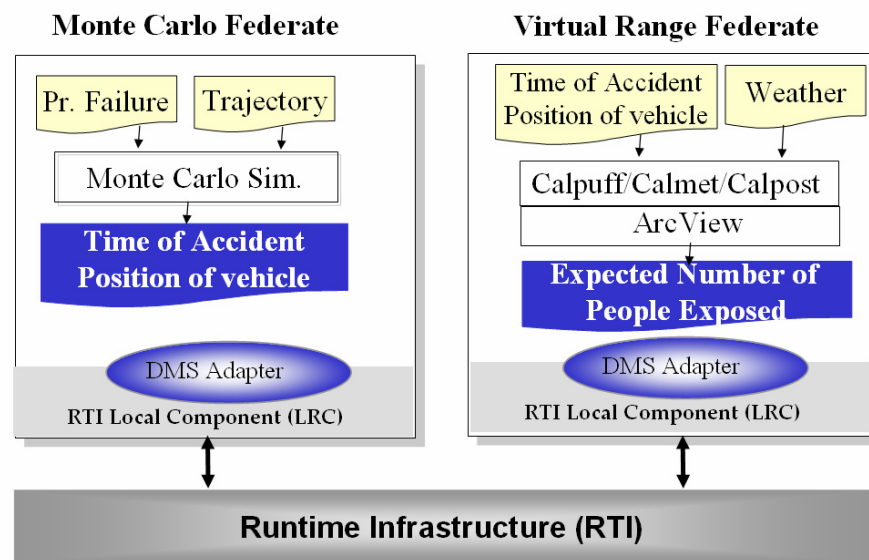


Figure 34 The MonteCarlo Federate and The VR Federate architecture

The advantages of this adaptation are twofold. First, we can simulate many different shuttles and orbiters which may have different probability of failure without changing the Virtual Range toxicity model. Since the inputs to the Monte Carlo federate highly depend on the property of the shuttle, each Monte Carlo federate can be built such a way that a Monte Carlo federate represents for a shuttle as a components model. The component model can be integrated to the VTB as it is necessary. Second, the process of each federate can be initiated by other simula-

tions such as the LaunchPad federate or the Mission Control federate and the intermediate information can be utilized by other federates in the VTB.

Weather Expert System (WES)

The Web-based Weather Expert System (WES) is a critical module of the Virtual Test Bed development to support “GO/NO-GO” decisions for Space Shuttle operations in the intelligent Launch and Range Operations (ILRO) program of NASA: the description of which in this section references (Rajkumar & Bardina, 2003). The weather rules characterize certain aspects of the environment related to the launching or landing site, the time of the day or night, the pad or runway conditions, the mission durations, the runway equipment and the landing type. Expert system rules are derived from weather contingency rules, which were developed over several years by NASA. Backward chaining, a goal-directed inference method, is adopted to the system rules, because a particular consequence or goal clause is evaluated first, and then chained backward through the rules. Once a rule is satisfied or true, then that particular rule is fired and the decision is expressed. The expert system is continuously verifying the rules against the past one-hour weather conditions and the decisions are made.

The launch weather guidelines/factors involving the Space vehicles, which are used as rules for the weather expert system, are similar in many areas, but distinctions are made for the particular characteristics of each. These guidelines are very conservative and seek to avoid possibly adverse conditions that focus on ambient temperature, wind speed, precipitation, lightning, type of clouds, cloud temperature and thickness.

The virtual test bed is used to simulate the mission, control, ground-vehicle, launch and range operations, during which weather plays a crucial role. Complex operations and their implications raised the need for an automated weather expert system to help analyze and provide expertise to management. The expert system's primary goal is to make its weather expertise available to macro level decision makers who need answers quickly and rapidly. It can help assess situations and facilitate launch planning. Analyzing text, numeric data and satellite images, the expert system helps save money by reducing the time involved in weather analysis allowing management to be more productive by making smarter, faster decisions.

In the weather expert system, the user interface is automated in such a way that the inputs to the expert system are downloaded and fed to the system in a periodic manner. There is no need for human intervention in the expert system and decisions for launch are automatically displayed as a web page. The weather web expert system is based on Java technology and web enabled, which can be viewed from any part of the world.

The real time weather data is obtained from different federal weather monitoring agencies. Images and other types of data are downloaded and then processed, extracted and converted to suitable numerical values. The image processed data is stored in an image file and other numerical values are stored in a weather file. The above mentioned files constitute the inputs for the expert system. NASA derives the rules for the weather expert system from weather contingency rules developed over several years. An example of a weather rule is (Rajkumar et al., 2003):

*If (36 < Temperature < 98) and
 (0 < Wind Speed < 24) and (Precipitation = "No") and
 (Lightning= "No") and (Cloud Temperature > 32) and
 (Cloud height > 20000 ft) and (Cloud thickness < 4500 ft) and
 (Cumulus = "No") and (Cumulonimbus = "No")
 Then
 Launch="GO"*

The above rule has nine antecedent clauses joined by a conjunction “and” and has a single consequent clause (Launch). The rule is triggered, if all antecedent clauses are set to be true. The clause conditions are derived for each vehicle type. Depending upon the launch vehicle, the rules are slightly changed. The rule variables remain constant for most of the launch vehicles. The rule base consists of rules for GO and NO-GO decisions. Depending upon the prevailing weather conditions, decisions are made. The advantage of the weather expert system (WES) is a unified decision of various weather factors which affects Shuttle launch.

We identified the weather expert system (WES) as an essential component in the Virtual Range. In order to integrate the WES to the VR, we decided to convert Socket based VR architecture (Sepulveda et al., 2004a) to the HLA based distributed simulation .

The integration of the WES into the VR infrastructure was accomplished using the RTI APIs exposed through the Java Binding. The WES can pass information about the decision and the different weather information into the VR and its federates. The integration of the WES into an HLA federation includes developing Federation Object Model (FOM) based on the information which needs to be exchanged, implementation of FederateAmbassador and adding some adaptation code into WES.

First, based on the objective of the federation, the shared information needed from the WES are mainly the launch decision and the processed data which are collected from various weather sources and then incorporated into the decision algorithm.

Second, the interface was implemented using RTI Java Binding which is a thin layer of C++ code that exposes the native C++ API of the RTI to Java applications through Java Native Interface (JNI) (DMSO, 2001). In addition, since the original WES is running on a Tomcat Web Server and the users interact with the application by using a web browser, it is difficult to make

WES as a federate within the current VTB architecture. Therefore, most of classes in the original WES were converted into Java Applications without altering the main algorithms, which is a simple change in the Java interface (Figure 35).

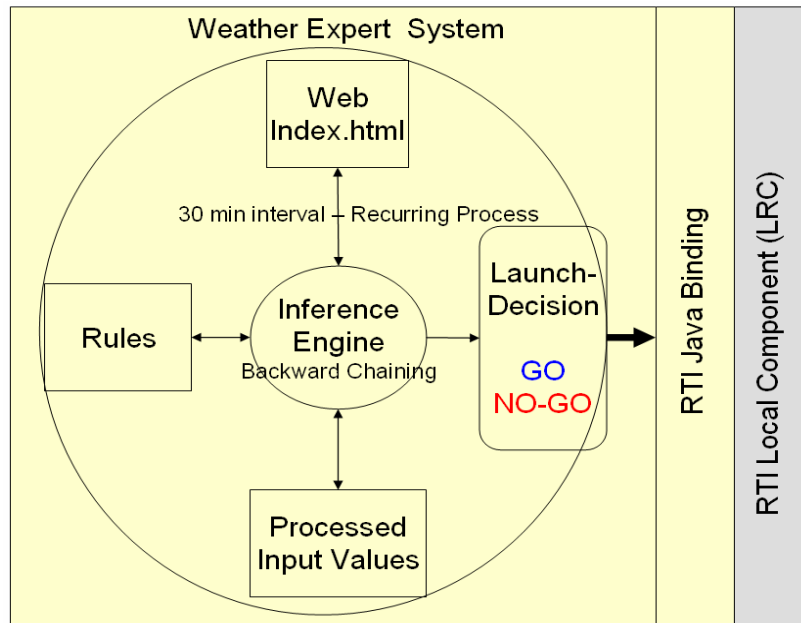


Figure 35 Adapted Weather Expert System (WES) Architecture to the HLA

In addition to the conversion, we have created a user interface to initiate the operations including acquire source data from various weather sources, process, invoke decision-making processes and joining to the VTB federation as a WES federate (Figure 36). The WES federates publishes (updates) near-real time weather information and GO or NO-GO decisions requested by the federation. The weather information and GO or NO-GO decisions will be used not only to the VR as a weather factors for CALPUFF models but also to Mission Control model to decide a launch weather criteria.

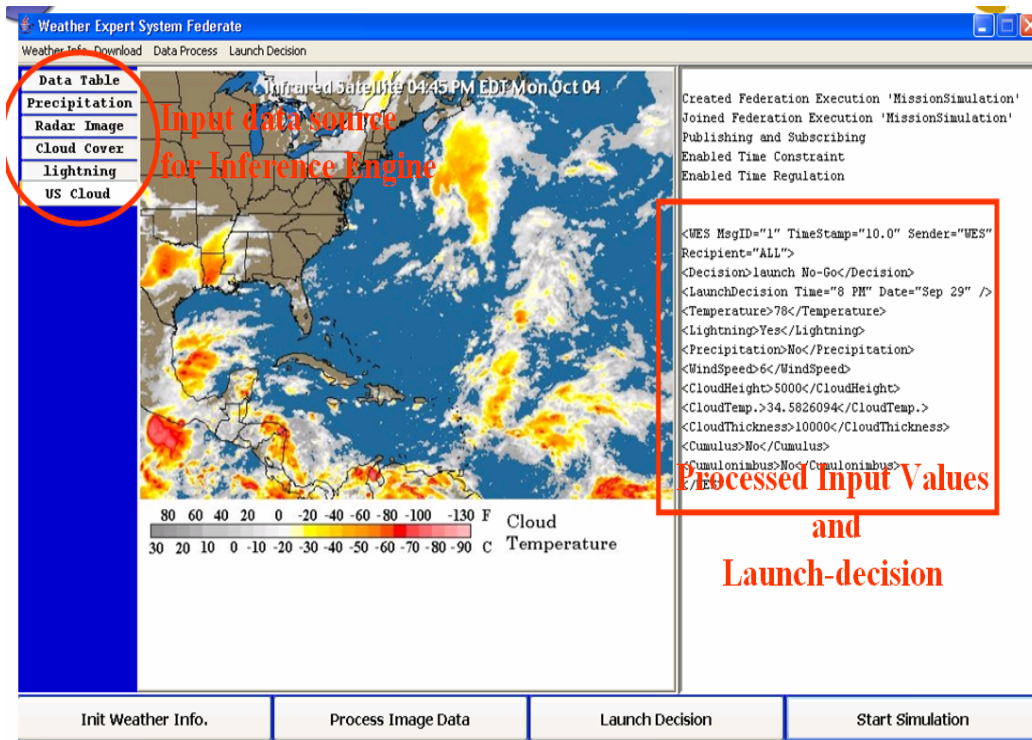


Figure 36 An implementation of the WES federate

Implementation of Space Shuttle Simulation Model using the SPEEDES Process Model

This section details a SPEEDES Process Model based implementation of Shuttle Transportation System (STS) model which was originally developed by Arena from Rockwell Software. This implementation will provide the following benefits. First, it makes the new model to be able to interact with the VTB federation through SPEEDES HLA gateway. Second, the new model will be flexible enough to extend external interfaces which may run asynchronously from it. Finally, the model may run on various computer architectures. Not to mention parallel execution, speed-up, and the advantages of Object-Oriented simulation modeling (Joines & Roberts, 1995; Rossetti et al., 2000; Law et al., 2000). These are benefits of the transfer and also the requirements of VTB we presented in Chapter 2. The needs for (selecting SPEEDES Process

Model as a modeling language for) transferring the original model into SPEEDES modeling framework is apparent for two reasons. First, the original model was developed in the COTS simulation package, Arena, that is actually an extended network of the basic modeling modules (or templates) concept of which are common to many other COTS simulation packages. In Arena process-oriented world view of discrete event simulation is applied for modeling simulation systems.

Second, by implementing a simulation model in SPEEDES architecture the model can include many exceptionally advanced features which are not available by other simulation languages or packages (e.g. parallel execution, multiple time management, load balancing, and external module interface, among others). The new model environment will prove even more useful with the future additions to the original simulation model. These additions will allow for different resolution levels and the study of safety and human-behavior modeling issues. The second reason using SPEEDES modeling framework is that it provides the HLA interoperability through its HLA gateway as described in Chapter 2. The development process includes the description of how the basic modeling elements the concept of which are common in COTS simulation packages, are facilitated and the process of coupling it to the HLA-RTI by SPEEDES' HLA Gateway.

SPEEDES' Process Modeling facilities and the basic modeling components classes were utilized to model the process (at high level) of the Shuttle Transportation System (STS). The source of the process flow (high-level) model was the NASA Shuttle Simulation Model. The NASA Shuttle Simulation Model is a simulation model for the operational life cycle of the Space Shuttle flight hardware elements through their respective ground facilities at KSC, and to on-orbit operations. The modeling approach of the NASA Shuttle Simulation Model was done at a

macro level, it included among others, the major processing facilities, ground support equipment, and flight hardware elements.

The basic modeling element classes described in Chapter 2 are utilized in the transfer process. The flight hardware elements are modeled as active entity classes (subclass of S_Entity), each of which has its own lifecycle of activities. The classes include the orbiter, main engines, the left and right orbiter maneuvering system pods, the forward reaction control system, the solid rocket boosters, and the external tank, among many others. The supporting facilities for the major flight hardware elements are modeled also as active entity classes (subclass of S_Process) each of which have different service time, schedule and require various resources.

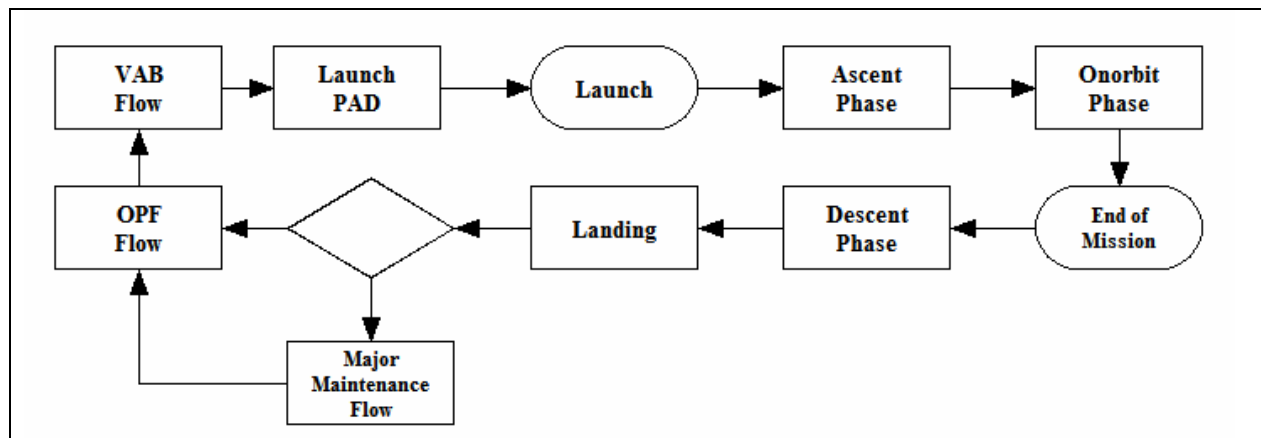


Figure 37 Life cycle of the Orbiter (as an active entity)

The classes include Orbiter Processing Facility, Vehicle Assembly Building, Launch Pad, Assembly Refurbishment Facility, and Main Engine Processing Facility, among many others. Ground support equipment is modeled as a passive entity class (subclass of S_Resource), each of which has its own schedule and capacity. The model logic which includes the process of classes, the stochastic input data, the hierarchy of classes (orbiter), and business rules follows that of (Cates et al., 2002). Figure 37 shows the life cycle of the main active entity in the model: the or-

biter. Figure 31 shows a code sample of the S_Orbiter class definition on SPEEDES Process Model.

```
void S_Orbiter::Process() {
P_VAR;
P_LV(double, delay_start_VAB);
    . // more P_LV (type, variable name)
    .
int successFlag;
SpObjHandle ProcessMgrHandle;

P_BEGIN(11);
delay_start_VAB = rand->GenerateDouble(10.0, 25.0);
WAIT(1, delay_start_VAB);

// ----- For VAB -----
SCHEDULE_VABQueue_Enqueue(SpGetTime(), SpGetObjHandle
    ("S_VABQueue_MGR",0), SpGetObjHandle());
WAIT_FOR_RESOURCE(2, resource_VAB, 1, -1, successFlag);

processTime_VAB = rand->GenerateDouble(150.0, 200.0);
WAIT(3, processTime_VAB);

SCHEDULE_VAB_Release(SpGetTime(), SpGetObjHandle("S_VAB_MGR", 0),
    *(SpObjHandle*)(void*)process_VAB);

delay_VAB_LaunchPad = rand->GenerateDouble(250.0, 500.0);
WAIT(4, delay_VAB_LaunchPad);

// ----- For LaunchPad -----
SCHEDULE_LaunchPadQueue_Enqueue(SpGetTime(), SpGetObjHandle
    ("S_LaunchPadQueue_MGR",0), SpGetObjHandle());
WAIT_FOR_RESOURCE(5, resource_LaunchPad, 1, -1, successFlag);

processTime_LaunchPad = rand->GenerateDouble(100.0, 120.0);
WAIT(6, processTime_LaunchPad);

// ----- Request for Launch -----
SCHEDULE_Orbiter_SendMessage(SpGetTime(), GetObjHandle(),
    "Request:Launch", "S_Orbiter", SpGetTime(), 0.0);
WAIT_FOR(7, launchapp, -1);
launchapp.Unset();
// ----- Dispose -----
SCHEDULE_Orbiter_Dispose(SpGetTime(), SpGetObjHandle());
P_END;
}
```

Figure 38 S_Orbiter Process (Active Entity) Logic Example

As shown in the simplified code example in Figure 32, the Process Model with the addition of basic discrete-event simulation classes supports a readable structure for modeling the active entities which in general represent the logical structure of the system being modeled.

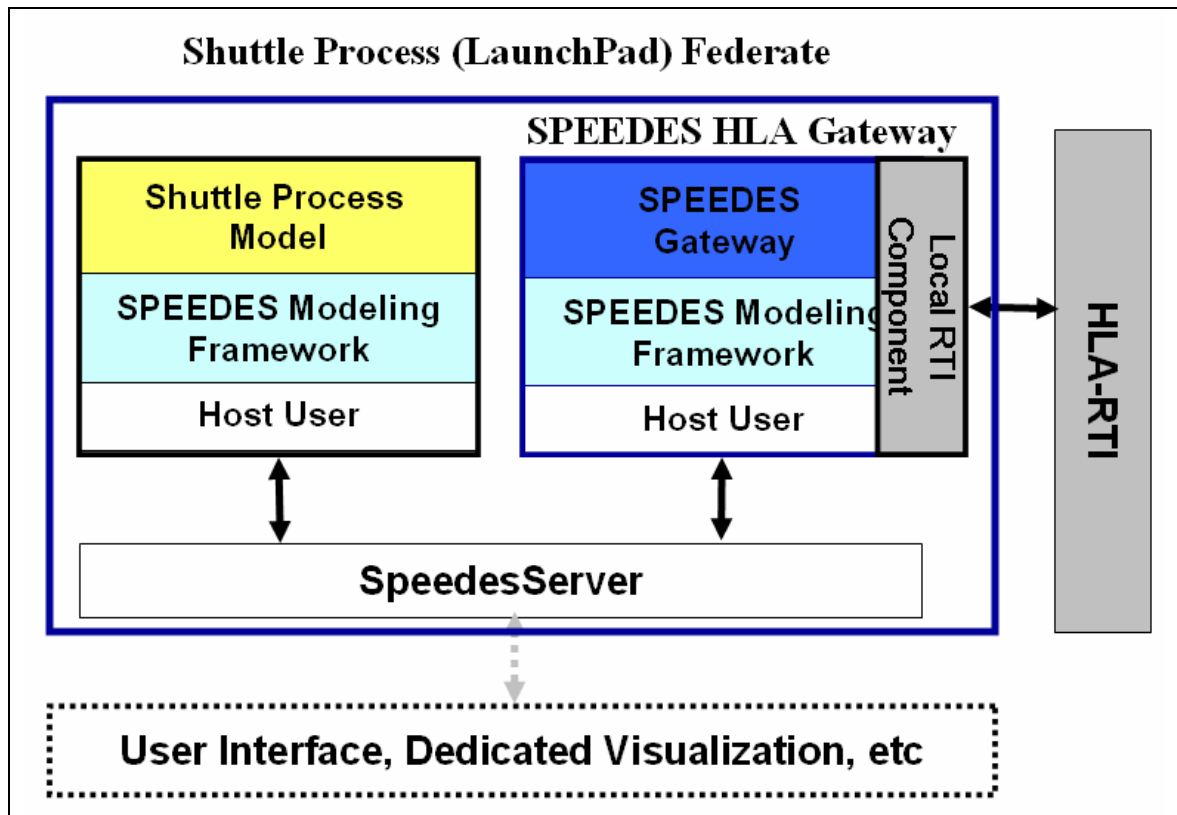


Figure 39 Space Shuttle Federate Architectue

The Space Shuttle federate consists of three executable applications: (1) Shuttle Process Model which is a pure SPEEDES simulation model built incorporating the process model classes, (2) SpeedesServer which enables communication between the SPEEDES model and the external modules by the `HostRouter` interface, (3) SPEEDES HLA Gateway which consists of the RTI interfaces implementation (`FederateAmbassador` and `RTIambassador`), a “bridge” coordinating two-way flow of information from both the RTI and `SpeedesServer`. In addition to the RTI interface in Shuttle Process federate, since Shuttle Process model is implemented such a way that it passes informa-

tion through SpeedesServer, some additional applications such as a User Interface and a dedicated visualization tools can be implemented via SpeedesServer.

Integration of the Framework for Spaceport Simulation System

This section details a prototype implementation of the VTB simulation system for a proof of concept experiment. The models integrated into the VTB simulation system represent different systems (simulation system and live information system) that interact in the simulation of liftoff. These models are the key simulation models that are currently available to VTB team in some form of simulation software and identified as models which represent distinctive traits of the VTB environment at the operation level, and which are independently developed (or transfer to), in different times and with different purposes. The models include the Launch Pad federate, Mission Control federate, Weather Expert System federate, Monte Carlo federate, and Virtual Range federate. Three major simulation modeling tools – SPEEDES, Arena, and AnyLogic – as well as non-simulation tools such as ArcMap and Calpuff are incorporated to created the models and for coupling of the models into the VTB distributed simulation environment, SPEEDES HLA Gateway, AnyLogic’s HLA Support Module (HSM), and DMS Adapter are utilized. The VTB federation leverages the existing models and uses HLA-RTI interfaces as a supporting distributed simulation engine for exchanging information between the models. The HLA provides a standard mechanism for interoperability and integration of simulations and supporting non-simulation tools with respect to a means of communication and a standard representation of common data. Figure 13 shows the configuration of the prototype implementation of VTB simulation system.

The configuration of the prototype is described in parts. First, a brief functional description of each model included in the prototype implementation is presented, and then the interactions between the models during the federation execution are described.

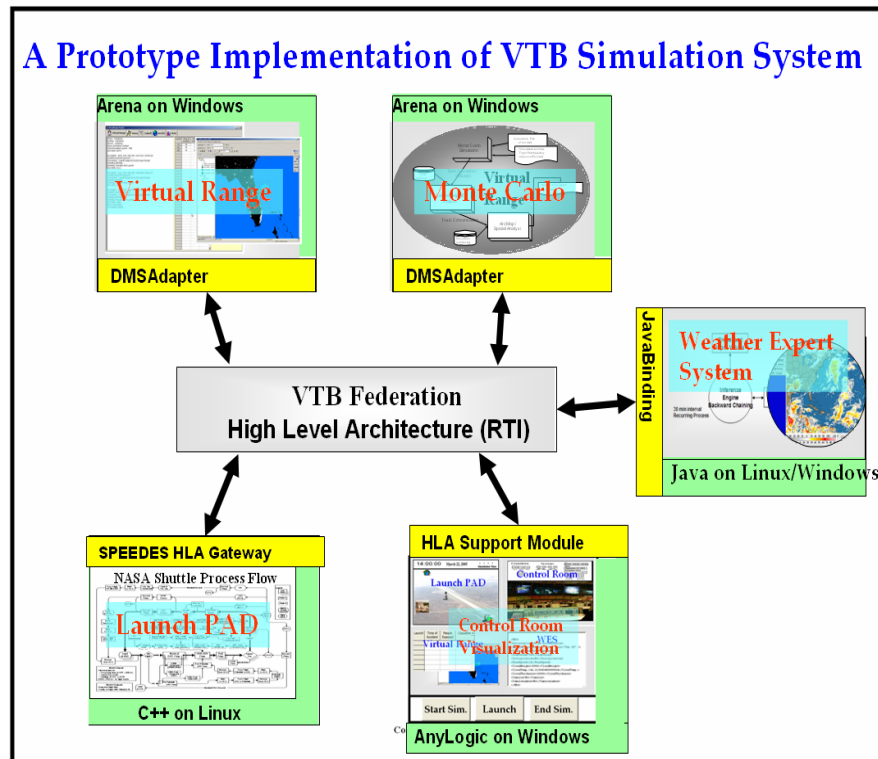


Figure 40 A Prototype Implementation of VTB Simulation System

Interactions Between the Federates

This section presents a brief functional description and the interactions of each model taking place during the federation execution of the prototype implementation (Figure 41).

LaunchPad Federate

The LaunchPad Federate is an adaptation of the NASA Space Shuttle Processing Model

(Cates et al., 2002) that simulates the flow of a space shuttle from landing at KSC, through its normal OPF flow, the VAB SSV flow, and its pad flow.

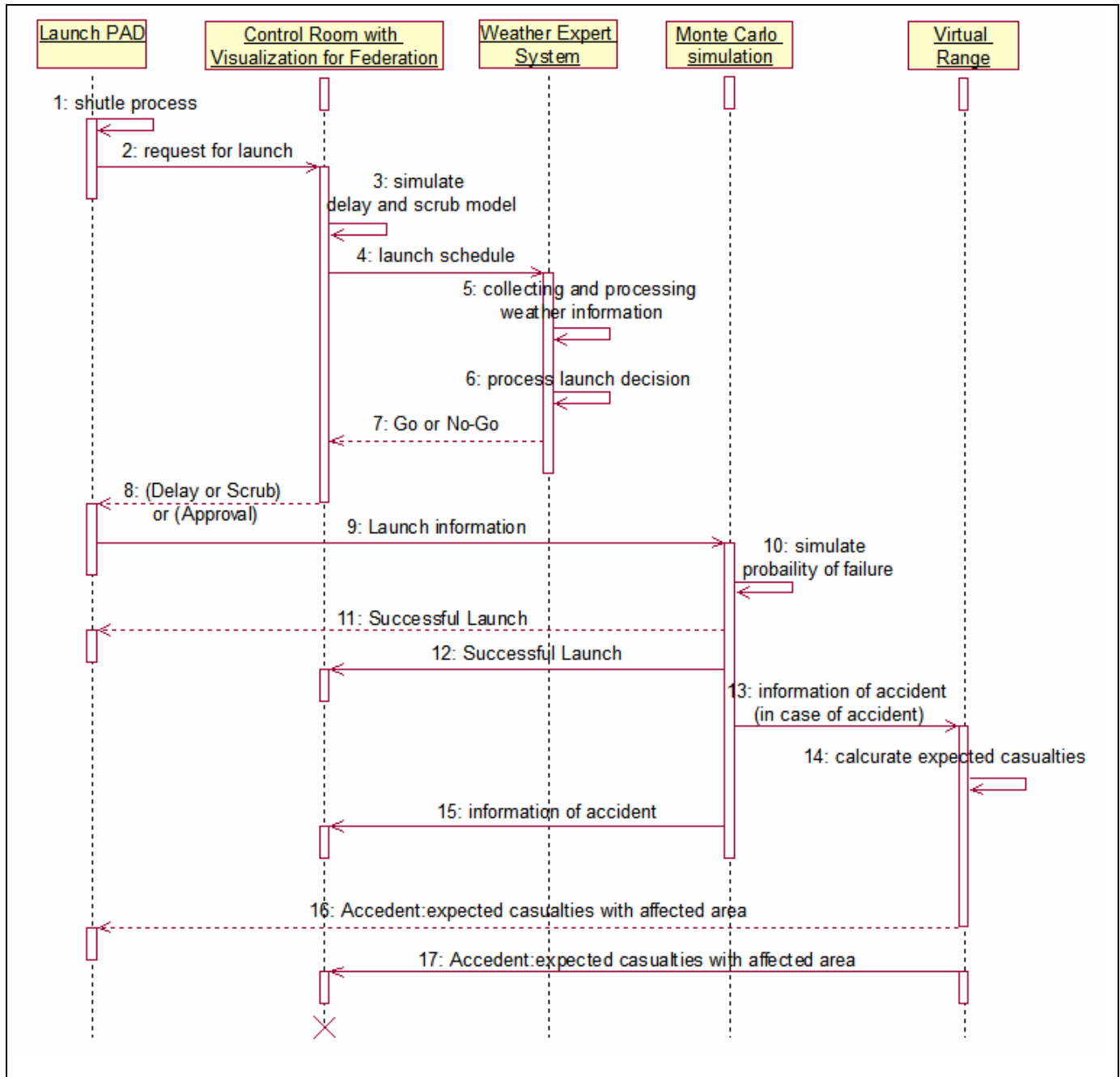


Figure 41 Interactions between the models during the federation execution

The simulation starts with the Shuttle reaching the pad in the LaunchPad federate and sending a signal through the RTI indicating that it is ready to launch. In the LaunchPad federate, when the shuttle arrives at the pad, a message is sent to the Mission Control federate and the orbiter waits for authorization to launch. It is possible that the Launch Pad federate may get a number of delay or scrub messages before getting authorization. When the authorization arrives, the Shuttle lifts off and at the same time a launch message is sent to Monte Carlo federate, and then waits for message either “successful launch” from Monte Carlo federate or “accident” with number of expected casualties from Virtual Range federate. In the case of “successful launch”, the LaunchPad federate displays flying around Earth and later returning to KSC, landing, and going through the cycle again. If the mission ends up in an accident, the LaunchPad federate changes the screen and shows the shuttle exploding, the date and time of the accident, the coordinates where the explosion occurs, and the amount of contaminant (from the Shuttle’s unused propellants) released into the atmosphere at that point.

Visualization (Mission Control Federate: The Launch Delay and Scrub Model)

The Mission Control federate evaluates the range and after getting through the RTI, the go ahead from the Weather federate, eventually authorizes the launch. This federate is activated when it receives a message from the LaunchPad federate that the orbiter is at the pad and ready to launch. The Mission Control federate then checks the systems in order to launch. The Mission Control federate checks for any failures within the four systems. After checking the four systems and verifying that no delays or scrubs occur, and also getting a “GO” launch-decision from the

Weather Expert System federate for the launch schedule, a message is sent through the RTI that the systems in the Mission Control federate are all green and the launch is a GO.

The Mission Control federate is the place where all the critical decisions are made during the launch operation. It is desired to make as much information as possible available to the federate in a timely manner in order to support decision making process. For this purpose we augmented the visualization of the VTB distributed simulation into the federation and the Mission Control federate. The visualization includes the state of all five federates in the federation as well as the messages passing between the federates. Since one of the main uses of visualization is to provide the system state to the user for interacting with the simulation system, we created command buttons on the screen to demonstrate a primitive interaction to the federation.

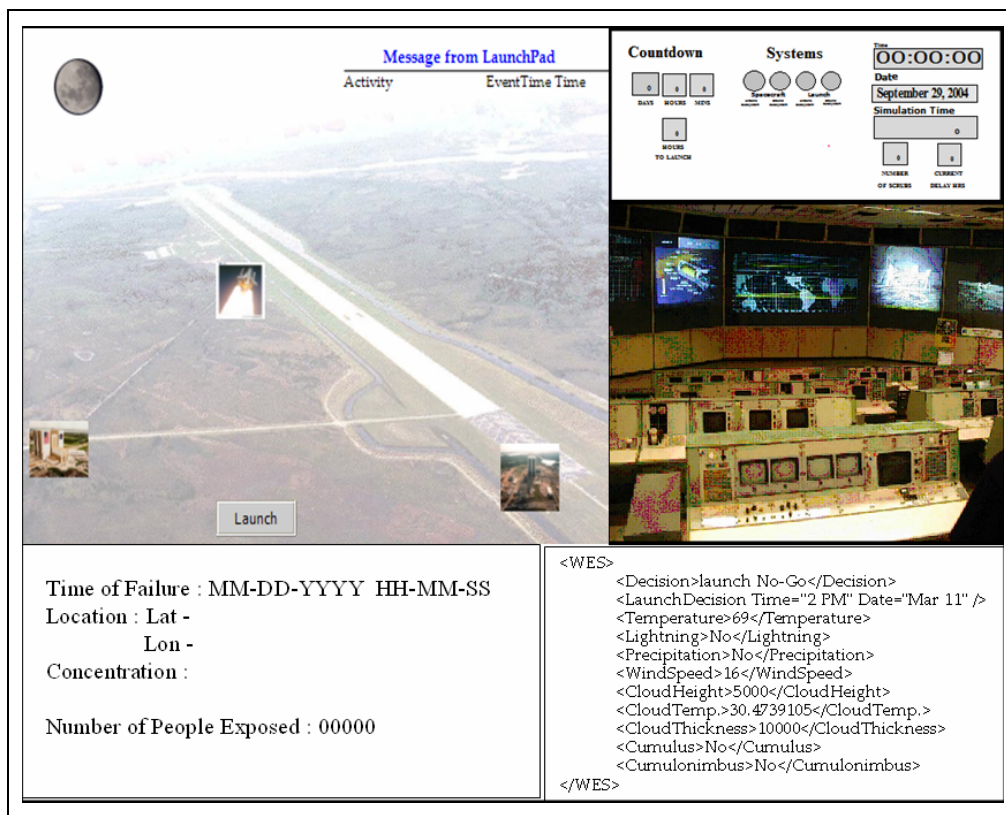


Figure 42 Visualization of Mission Control Room

Monte Carlo Federate

When the simulated orbiter lifts off, the Monte Carlo federate is notified through the RTI. The Monte Carlo federate model receives the message from the Launch Pad federate that a launch took place. The Monte Carlo federate then determines whether the mission is successful or whether a simulated disaster occurs. This is done (as in all Monte Carlo simulations) through the generation of a random number to decide whether there is going to be an accident or if there is going to be a successful launch. If there is a successful launch the Monte Carlo federate sends messages through the RTI to the Launch Pad federate and the Mission Control federate; if there is an accident it sends messages which include the location (latitude, longitude, and altitude) and the amount of toxicant released to the Virtual Range federate and the Mission Control federate.

Virtual Range Federate:

If the message from the Monte Carlo federate indicates a successful launch, the Virtual Range federate displays a counter of the number of launches and shows a summary of the current weather. As with all other Federates, the Virtual Range federate includes a clock showing date and time. If the message from the Monte Carlo federate indicates an accident, the Virtual Range model is activated and extracts the location of the accident in space and the amount of contaminants released in the atmosphere from the message. In the Virtual Range model, Calpuff is then initiated and uses the weather conditions for the day of the simulated launch to determine the concentration of the contaminant in different locations around the accident site one hour after the accident. This information is then input as a layer in ArcMap and the points where the concentration of the pollutant exceeds the limits determined by the contaminant's Exposure Response

Curves are displayed over a map of Florida. Spatial Analyst, a companion software to ArcMap, performs a query on population exposed over the highlighted area. The population data is taken from LandScan and imported as a separate layer into ArcMap. At the end, the Virtual Range federate reports the number of people exposed to toxic levels of the released toxic propellant by showing a map of Florida with the area effected by the accident.

Weather Expert System (WES) federate

The Weather Expert System (WES) is a Java-based model. It shows a summary of the weather forecast (updated each day). It collects the information from different websites; for example, the temperature and wind speed from <http://weather.noaa.gov/weather>. When the WES Federate receives the message that the Mission Control systems are a GO, it checks whether the weather conditions are also a GO. If so, the message is sent through the RTI indicating that the launch is authorized.

CHAPTER SIX: CONCLUSION

Summary

As the size, complexity, and functionality of systems we need to model and simulate continue to increase, benefits such as interoperability and reusability enabled by distributed discrete-event simulation are of enormous interest in many disciplines. This research proposes a distributed simulation based framework for modeling and simulation of complex systems. The framework overlooks simple objects and components and views the complexity of a simulated system at a system-wide level. A simulation of a complex system which is itself a system of systems can be managed by aggregating individual subsystems within the framework.

The distributed simulation engine used with this framework is the High Level Architecture (HLA). The legacy simulation models we have built in previous projects with the Virtual Range Model and the Weather Expert System are integrated into the framework.

An important aspect of the approach to modeling complex systems adapted in this research effort is that any model developed using COTS simulation languages with HLA interoperability such as Arena and AnyLogic, or general purpose programming languages such as C++ and Java can be used to model complex systems.

In the case study of modeling Spaceport, the framework has been designed as a distributed simulation to facilitate the integrated execution of different simulations, (shuttle process model, Monte Carlo model, Delay and Scrub Model) each of which is addressing different mission components as well as other non-simulation applications (Weather Expert System and Virtual Range). Although these models were developed independently at various times, the original

purposes and organizations have been seamlessly integrated, and interact each other through the RTI to simulate a shuttle launch related processes.

In order to support a seamless integration of essential simulation components in Spaceport, we have presented the HLA, a state of art distributed simulation engine, currently available, with multiple approaches of adaptation to Spaceport requirement. These adaptation approaches include (1) designing the primitive simulation component classes to promote an easier model building process and to support shared data structure compatibility within the federation, (2) customizing existing simulation models to be able to interact each other through the HLA-RTI with help of *DMS Adapter*, *SPEEDES HLA Gateway*, and *AnyLogic's HLA Support Module*, and (3) developing a visualization method that animates the state of a remote simulation in the local federate using COTS simulation packages with HLA interoperability support. This visualization can be seen as a prototyping method of the life cycle model. The method of visualization of remote federates presented in this study can be used to identify user requirements and the level of fidelity required, and test user interface design. After identifying properties of interest through prototyping the visualization can be developed in high resolution and even in a 3D or virtual reality environment.

While the framework is implemented in the context of Spaceport simulation, it is devoted to address a small number of subsystems in the complex system, which is actually a simplified representation of Spaceport. The same modeling framework developed in the study could easily be ported to various disciplines such as distributed manufacturing, supply chain, and enterprise engineering.

This study found that the defining properties of complex systems - interaction and emergence – are realized and the software life cycle models (including prototyping and the spiral

model) are used as metaphors, not actual development processes, to manage the complexity of modeling and simulation of the system. The system of systems continuously evolves to accomplish its goals, during the evolution subsystems coordinate with one another and adapt with environmental factors such as policies, requirements, and objectives. In addition, when the prototyping of experimental models using COTS simulation languages proves its feasibility to address the problem, high fidelity, high performance modeling can be considered. The prototyping method is applicable because COTS simulation languages are more powerful and easy to use and are inexpensive. For these reasons, COTS simulation languages are becoming more commonplace and are often equipped with useful pre-built modules (or templates) and a relatively high quality animation.

Limitation

There are a number of challenges remaining to make this framework work systematically with respect to the integration of the models into the application repository which comprises of simulation models and non-simulation software. Although the current framework proves the feasibility of interoperability, it requires many steps to realize the how the models interact. The steps include coupling an interface to the standard distributed simulation engine, developing common data model, and adapting existing simulation code into a new distributed simulation environment, among others. Model integration at the software level requires a considerable amount of work if they are developed in different modeling languages.

A variety of COTS simulation languages have been developed and used widely in many disciplines, not many of which provide an interface to the standard distributed simulation. Since

the framework is based on the distributed simulation interfaces must be developed either by the COTS simulation language vendor or the simulation developer. If the interface is embedded in the COTS simulation language's framework it may reduce the complexity of integration in many levels. However, if we can develop a gateway which enables COTS simulation languages to interact within the distributed simulation it is also acceptable and reduces complexity to some degree.

The number of models in the repository for a system of systems needs to be extended to achieve the objective of complex systems. In the case of Spaceport cost models, training models, 3D real-time visualization of range, and optimization models of mission-related operations would be some examples.

Future Work

While extensive research has been performed to develop a framework for modeling complex systems, much work still remains. As stated in the previous section, some limitations exist in multiple areas. To enhance applicability of the framework to a wide range of complex system simulation development, the framework can be extended in several areas.

Model Driven Architecture

The most immediate need is for a repository of the simulation models which covers many simulated systems in Spaceport and in VTB environments. The individual model should be reusable. Some important factors in simulation model reuse are: proper documentation, unambiguous interface, selection of proper COTS simulation languages which are flexible, adaptable and ex-

tensible enough. The advantages from reuse of existing models would be: (1) low cost of modeling and rapid development; (2) the quality of reused model increases because it will be examined by continued reuse; (3) it may reduce the complexity of the modeling process (Crnkovic, 2004; Lau, 2004; Atkinson, 2002; Whitehead, 2002).

There is a relatively new technology called Model Driven Architecture (MDA) which “provides an approach for designing and building component-based systems that remain decoupled from languages, platforms and middleware environment that are eventually used to implement the system”. The core concept of MDA is the model, which uses industry standards like Unified modeling Language (UML) and Meta-Object Facility (MOF) to notate and store the system. The model is the key artifact in an MDA system and remains central throughout design and development. The model is independent of the eventual platforms and service used to implement the system (Bing, Hongji, Chu, & Baowen, 2003; Ramljak, Puksec, Huljenic, Koncar, & Simic, 2003; Gracanin, Bohner, & Hinchey, 2004; Uhl, 2003). If the simulation models in Spaceport are designed based on MDA and then stored in the repository, the model can be transitioned and then executed in the selected computing environment.

Adapting Web-based Simulation

Although the framework is based on distributed simulation, the operating environment of a model is very limited to the specific configuration of system as the model is developed. This also limits the accessibility of models; in order to address this type of problem a possible future work would be adapting Web-based simulation into the framework. The integration of Web technology and distributed simulation technology will provide many methods to extend the

availability of simulation models. There are multiple technologies that exist such as Remote Method Invocation (RMI) from Sunsoft' Java Development Kit (JDK), Common Object Broker (CORBA) by the Object Management Group (OMG), etc. that could be used to extend the availability of simulation models. RMI enables the programmer to create distributed Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines. RMI has its own native Object Request Broker (ORB) and eliminates the need to write an IDL (Interface Definition language). One limitation of RMI is that the development of application is limited within the Java language. The CORBA technology supports a standardized framework to support application development and interoperation in a distributed and heterogeneous environment by separating interfaces from object implementations. The main concept of the CORBA approach is that client and server are isolated by a well defined interface which allows the client to access server functionality without knowing the underlying transport/protocol or server's implementation details (Dang Gang & Jin, 2000; Page et al., 2000; Buss & Jackson, 1998).

Integrating dedicated Visualization tools to the framework

Additional work is also needed to integrate a high fidelity visualization model into the framework. One of the most compelling components in complex simulation system would be the visualization. Integrating real-time high-performance visualization in Spaceport will help the user to better understand the simulated operation and interact with the simulation system more effectively; especially when the goal of simulation is training the immersive high fidelity visualization is a must have. We have surveyed some of existing dedicated visualization in the domain

of space operations (Compton et al., 2003). The proper visualization model for Spaceport would be a 3-D visual representation of the facilities required to perform the functions for space transportation systems. Its objective is to immerse the space transportation systems engineers in their domain, to discover the infrastructure and operations implications, across the systems life cycle from different perspectives. It complements the analytical tools provided by the core model which are linked to data sheets for cost and cycle time information that support all the different functions (McCleskey Carey M., 2001).

LIST OF REFERENCES

- AIAA (1991). *Atmospheric Effects of Chemical Rocket Propulsion*. New York: American Institute of Aeronautics and Astronautics (AIAA).
- Arief, L. B. & Speirs, N. A. (2000). A UML tool for an automatic generation of simulation programs. *Proceedings Second International Workshop on Software and Performance. WOSP2000*, 71-76.
- AST (2002). Operating in the Federal Ranges, Student Training Guide. Safety Inspector Training and Qualification Program, Washington [On-line].
- Atkinson, C. (2002). *Component-based product line engineering with UML*. London: Addison-Wesley.
- Bailey, C. A., McGraw, R. M., Steinman, J. S., & Wong, J. (2001). SPEEDES: a brief overview. *Proceedings of the SPIE - The International Society for Optical Engineering*, 4367, 190-201.
- Bapat, V. & Sturrock, D. T. (2003). The Arena Product Family: enterprise modeling solutions. *Proceedings of the 2003 Winter Simulation Conference*, 210-217.
- Bardina, J. (2001). Intelligent Launch & Range Operations. *NASA ARC*.
- Barth, T (2002, April). Found in Space. *IIE Solutions*.
- Bing, Q., Hongji, Y., Chu, W. C., & Baowen, X. (2003). Bridging legacy systems to model driven architecture. In (pp. 304-309).
- Blocher, T. W. (2002). Information visualization in a distributed virtual decision support environment. *Proceedings of the SPIE - The International Society for Optical Engineering*, 4716, 323-329.
- Boccara, N. (2004). *Modeling complex systems*. New York: Springer.
- Boer, C. A. & Verbraeck, A. (2003). Distributed simulation with COTS simulation packages. *Proceedings of the 2003 Winter Simulation Conference*, 829-837.
- Borshchev, A., Karpov, Y., & Kharitonov, V. (2002). Distributed simulation of hybrid systems with AnyLogic and HLA. *Future Generation Computer Systems*, 18, 829-839.
- Braude, E. J. (1998). Towards a standard class framework for discrete event simulation. In (pp. 4-8).
- Buss, A. & Jackson, L. (1998). Distributed simulation modeling: a comparison of HLA, CORBA, and RMI. *1998 Winter Simulation Conference. Proceedings*, 819-825.

- CALPUFF Modeling System (2004). CALPUFF Modeling System.
<http://earthtec.vwh.net/download/download.htm> [On-line].
- Cates, G. R., Steele, M. J., Mollaghasemi, M., & Rabadi, G. (2002). Modeling the space shuttle. *Proceedings of the 2002 Winter Simulation Conference*, 754-762.
- Charles, M. & Frank, R. (2000). Simulation in the international IMS MISSION project: the IMS MISSION architecture for distributed manufacturing simulation. *2000 Winter Simulation Conference Proceedings*, 1539-1548.
- Committee on Space Launch Range Safety, A. a. S. E. B. N. R. C. (2000). *Streamlining Space Launch Range Safety* Washington, DC: NATIONAL ACADEMY PRESS.
- Compton, J., Sepulveda, J., & Rabelo, L. C. (2003). *Integration of the Virtual Test Bed and Virtual Range Assessment*.
- Crnkovic, I. (2004). *Component-based software engineering 7th international symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004 : proceedings*. Berlin: Springer-Verlag.
- Dang Gang, W. X. Z. W. & Jin, S. (2000). A prototype of Web-based distributed simulation environment. *Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, 732-737.
- DMSO. (1998a). Hands-On Practicum. Defense Modeling and Simulation Office .
 Ref Type: Slide
- DMSO (1998b). High Level Architecture Run-Time Infrastructure Programmer's Guide 1.3 Version 5. Defense Modeling and Simulation Office [On-line]. Available:
<https://www.dmsomil/public/>
- DMSO (1998c). HLA: Interface Specification. Version 1.3. Defense Modeling and Simulation Office [On-line]. Available: <https://www.dmsomil/public/>
- DMSO (1998d). HLA: Object Model Template Specification Version 1.3. Defense Modeling and Simulation Office [On-line]. Available: <https://www.dmsomil/public/>
- DMSO (1998e). HLA: Rules Version 1.3. Defense Modeling and Simulation Office [On-line]. Available: <https://www.dmsomil/public/>
- DMSO (2001). The RTI1.3-NG Java Binding. DMSO [On-line]. Available:
<https://www.dmsomil/public/>
- Earth Tech (2000). *A User's Guide for the CALPUFF Dispersion Model (Version 5)*. Earth Tech Inc.

- Fragola, J. & G. Maggio (1995). *Probabilistic Risk Assessment of the Space Shuttle. Phase 3: A Study of Potential of Losing the Vehicle During Nominal Operation, Vol. 2: Integrated Loss of Vehicle Model* SAIC (Science Applications International Corporation) to NASA.
- Fujimoto, R. M. (2003). Distributed simulation systems. *2003 Winter Simulation Conference Proceedings, 1*, 124-134.
- Fujimoto, R. M. (2000). *Parallel and distribution simulation systems*. New York: Wiley.
- Fullford, D. (1999). A Federation Management Tool: Using the Management Object Model (MOM) to Manage, Monitor, and Control an HLA Federation. *Spring Simulation Interoperability Workshop*.
- Gesser R. (2003). Practical Applications of Overwater Data to CALMET modeling on Coastal Domains. Atlanta, GA, Trinity Consultants.
Ref Type: Generic
- Ghosh, S. & Lee, T. S. (2000). *Modeling and asynchronous distributed simulation analyzing complex systems*. New York: IEEE Press.
- Gracanin, D., Bohner, S. A., & Hinchey, M. (2004). Towards a model-driven architecture for autonomic systems. In (pp. 500-505).
- Hanna, J. P. & Hillman, R. G. (2002). SPEEDES for distributed information enterprise modeling. *Proceedings of the SPIE - The International Society for Optical Engineering, 4716*, 160-166.
- Hill Brothers Chemical Company (2001). Material Safety Data Sheet.
<http://hillbrothers.com/msds/pdf/hydrochloric-acid-solution.pdf> [On-line].
- Johnson, G. D. (1999). Networked simulation with HLA and MODSIM III. *WSC'99.1999 Winter Simulation Conference Proceedings. 'Simulation - A Bridge to the Future'*, 1065-1070.
- Joines, J. A. & Roberts, S. D. (1995). Design of object-oriented simulations in C++. *1995 Winter Simulation Conference Proceedings*, 82-89.
- Judith, D., Richard, M. F., & Richard, M. W. (1998). The DoD high level architecture: an update. *1998 Winter Simulation Conference. Proceedings*, 797-804.
- Klein, U., Schulze, T., & Strassburger, S. (1998). Traffic simulation based on the High Level Architecture. *1998 Winter Simulation Conference. Proceedings*, 1095-1103.
- Kossiakoff, A. & Sweet, W. N. (2003). *Systems engineering principles and practice*. Hoboken, N.J: J. Wiley.
- Kuhl, F. & Riddick, F. (2000). Distributed Manufacturing Simulation Adapter.
Ref Type: Slide

- Kuhl, F., Weatherly, R., & Dahmann, J. (2000). *Creating computer simulation systems : an introduction to the high level architecture*. Upper Saddle River, NJ: Prentice Hall PTR.
- LandScan (2003). LandScan. <http://www.ornl.gov/sci/gist/landscan/index.html> [On-line].
- Lau, K. K. (2004). *Component-based software development case studies*. New Jersey: World Scientific.
- Law, A. M. & Kelton, W. D. (2000). *Simulation modeling and analysis*. (3rd ed ed.) Boston: McGraw-Hill.
- Lebo, D. & Woltman, M. (2002). *EIN5117 Final Report* University of Central Florida.
- Lin, J. T., Yeh, K. C., & Sheu, L. C. (1992). A framework for designing an animated simulation system based on model-animotor-scheduler paradigm. *1992 Winter Simulation Conference Proceedings*, 756-763.
- McCleskey Carey M. (2001). VISION SPACEPORT.
http://science.ksc.nasa.gov/shuttle/nexgen/Nexgen_Downloads/Vision_Spaceport_Report_042701.pdf [On-line].
- McLean, C. & Riddick, F. (2000a). The IMS MISSION architecture for distributed manufacturing simulation. *2000 Winter Simulation Conference Proceedings*, 1539-1548.
- McLean, C. & Riddick, F. (2000b). Integration of manufacturing simulations using HLA. *Proceedings of the Military, Government and Aerospace Simulation Symposium*, 237-242.
- Metron, I. (2003). *SPEEDES, Users Guide 2003* . Metron Corporation, San Diego, California.
- Nahum, G., Stephen, G. E., & Stuart, C. (1998). Information visualization. *interactions.*, 5, 9-15.
- NASA (2004). Follow NASA as we Explore: Earth, Moon, Mars and Beyond.
http://www.nasa.gov/missions/solarsystem/explore_main.html [On-line].
- NGS (2004). NGS -- the National Geodetic Survey NADCON, North American Datum Conversion Utility [Computer software]. NGS -- the National Geodetic Survey NADCON.
- NIMA (2001). The Universal Transverse Mercator (UTM) Grid, Fact Sheet 077-01.
<http://mac.usgs.gov/mac/isb/pubs/factsheets/fs07701.html#utm> [On-line].
- NIST. (2001). The Distributed Manufacturing Simulation Adapter Reference Guide. The National Institute of Standards and Technology (NIST).
Ref Type: Generic
- NIWAR (2004). *Good Practice Guide for Atmospheric Dispersion Modeling*. Wellington, New Zealand: National Institute of Water and Atmospheric Research (NIWAR).

- NOAA (2004). Unedited Surface Weather Observations. http://www.nndc.noaa.gov/cgi-bin/nndc/buyOL-001.cgi?FNC=qcall_Aswoqmain_hm [On-line].
- Page, E. H., Buss, A., Fishwick, P. A., Healy, K. J., Nance, R. E., & Paul, R. J. (2000). Web-based simulation: revolution or evolution? *ACM Transactions on Modeling and Computer Simulation*, 10, 3-17.
- Pawletta, S., Drewelow, W., & Pawletta, T. (2000). HLA-based simulation within an interactive engineering environment. In (pp. 97-102).
- Peter Lorenz (2003). Simulation and Animation . <http://isgwww.cs.uni-magdeburg.de/~pelo/s1e/sa1/sa1.shtml> [On-line].
- Pfleeger, S. L. (2001). *Software engineering theory and practice*. (2nd ed ed.) Upper Saddle River, N.J: Prentice Hall.
- Philipson, L. L. (1999). *An expert elicitation of estimates of exposure limits for space and missile launch toxicants Technical Report No. 99-400/11.2-01*.
- Pidd, M. (1998). *Computer simulation in management science*. (4th ed ed.) New York: John Wiley.
- Pressman, R. S. (2000). *Software engineering a practitioner's approach*. (5th ed ed.) Boston, Mass: McGraw Hill.
- Rabelo, L. (2002a). *SLS Document for the VTB NASA Kennedy Space Center*.
- Rabelo, L. (2002b). *Presentation to NASA Ames Research Center about the VTB*.
- Rabelo, L. (2002c). *The Virtual Test Bed Project: NASA Fellow 2002, RESEARCH REPORTS* John F. Kennedy Space Center.
- Rajkumar, T. & Bardina, J. E. (2003). Web-based weather expert system (WES) for Space Shuttle launch. *SMC'03 Conference Proceedings.2003 IEEE International Conference on Systems, Man and Cybernetics.Conference Theme - System Security and Assurance*, 5040-5045.
- Ramljak, D., Puksec, J., Huljenic, D., Koncar, M., & Simic, D. (2003). Building enterprise information system using model driven architecture on J2EE platform. In (pp. 521-526).
- Roberto, F. L., Guixiu, Q., & Charles, M. (2003). Manufacturing case studies: NIST XML simulation interface specification at Boeing: a case study. *Winter Simulation Conference*, 1230-1237.
- Rogers, P. & M.T.Flanagan (1991). On-line simulation for real-time scheduling of manufacturing systems. *Industrial Engineering*.

- Rohrer, M. W. (2000). Seeing is believing: the importance of visualization in manufacturing simulation. *2000 Winter Simulation Conference Proceedings*, 1211-1216.
- Rossetti, M. D., Aylor, B., Jacoby, R., Prorock, A., & White, A. (2000). Simfone': an object-oriented simulation framework. *2000 Winter Simulation Conference Proceedings*, 1855-1864.
- Rouse, W. B. (2003). Engineering complex systems: implications for research in systems engineering. *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 33, 154-156.
- Sage, A. P. & Olson, S. R. (2001). Modeling and Simulation in Systems Engineering: Whither Simulation Based Acquisition? *SIMULATION*, 76, 283-285.
- Sepulveda, J., Rabelo, L., Park, J., Gruber, F., & Martinez, O. (2004a). Factors Affecting the Expectation of Casualties in the Virtual Range Toxicity Model. In (pp. 685-692).
- Sepulveda, J., Rabelo, L., Park, J., Riddick, F., & Peaden, C. (2004b). Implementing the High Level Architecture in the Virtual Test Bed. In (pp. 380-387).
- Shishko, R., Aster, R., & Cassingham, R. C. (1995). *NASA systems engineering handbook*. Washington, D.C.: National Aeronautics and Space Administration.
- Sommerville, I. (2001). *Software engineering*. (6th ed ed.) Harlow, England: Addison-Wesley.
- Stafford, R. (1995). AutoView [model animation]. *1995 Winter Simulation Conference Proceedings*, 524-528.
- Steinman, J. S. (1998b). Time managed object proxies in SPEEDES. *Proceedings of Object-Oriented Simulation Conference (OOS'98). International Conference on Simulation and Multimedia in Engineering Education (ICSEE'98). 1998 Western MultiConference*, 59-65.
- Steinman, J. S. (1990). SPEEDES: synchronous parallel environment for emulation and discrete event simulation. *Advances in Parallel and Distributed Simulation. Proceedings of the SCS Multiconference*, 95-103.
- Steinman, J. S. (1998a). Scalable distributed military simulations using the SPEEDES object-oriented simulation framework. *Proceedings of Object-Oriented Simulation Conference (OOS'98). International Conference on Simulation and Multimedia in Engineering Education (ICSEE'98). 1998 Western MultiConference*, 3-23.
- Steinman, J. S., Berliner, G., Blank, G. E., Brutocao, J. S., Burckhardt, J., Peckham, M. et al. (1999). The SPEEDES-based run-time infrastructure for the high-level architecture on high-performance computers. *Proceedings of the High Performance Computing Symposium - HPC'99. 1999 Advanced Simulation Technologies Conference*, 255-266.

- Steinman, J. S. & Wong, J. W. (2003). The SPEEDES persistence framework and the standard simulation architecture. *Proceedings. Seventeenth Workshop on Parallel and Distributed Simulation*, 11-20.
- Steven D. Farr & Alex F. Sisti (1994). Visualization of General Purpose Simulation Results. <http://www.rl.af.mil/tech/papers/ModSim/DU94.html> [On-line].
- Strassburger, S. (1999). On the HLA-based coupling of simulation tools. *Modelling and Simulation: A Tool for the Next Millennium. 13th European Simulation Multiconference 1999. ESM'99*, 45-51.
- Strassburger, S. (2001). *Distributed Simulation Based on the High Level Architecture in Civilian Application Domains*.
- Strassburger, S., Schulze, T., Klein, U., & Henriksen, J. O. (1998). Internet-based simulation using off-the-shelf simulation tools and HLA. *1998 Winter Simulation Conference Proceedings*, 1669-1676.
- Swain, J. James (2003, August). Simulation Software Survey. *OR/MS Today*.
- Swider, C. L., Bauer, K. W., Jr., & Schuppe, T. F. (1994). The effective use of animation in simulation model validation. *1994 Winter Simulation Conference Proceedings*, 633-640.
- Thayer, R. H., Dorfman, M., & Christensen, M. J. (2002). *Software engineering*. (2nd ed ed.) Los Alamitos, Calif: IEEE Computer Society Press.
- Uhl, A. (2003). Model driven architecture is ready for prime time. *Software, IEEE*, 20, 70, 72.
- United Way of Brevard County (2002). LandScan. <http://sedac.ciesin.org/blue/gpw/landscan/> [On-line].
- Vliet, H. v. (2000). *Software engineering principles and practice*. (2nd ed ed.) Chichester England: John Wiley.
- Whitehead, K. (2002). *Component-based development principles and planning for business systems*. Boston, Mass: Addison-Wesley.
- Whitehurst, R. A. & Brutocao, J. (1998). Parallel execution of process-based simulation models. *Proceedings of Object-Oriented Simulation Conference (OOS'98). International Conference on Simulation and Multimedia in Engineering Education (ICSEE'98). 1998 Western MultiConference*, 115-120.