USING THE SOFTWARE ADAPTER TO CONNECT LEGACY SIMULATION MODEL TO THE RTI

By

Deepak Kumar R Rachapalli
B.Tech, JNTU University, July 2001

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Modeling and Simulation in the College of Engineering and Computer Science at the University of Central Florida Orlando, Florida

Spring 2006
ABSTRACT

The establishment of a network of persistent shared simulations depends on the presence of a robust standard for communicating state information between those simulations. The High Level Architecture (HLA) can serve as the basis for such a standard. While the HLA is architecture, not software, use of Run Time Infrastructure (RTI) software is required to support operations of a federation execution. The integration of RTI with existing simulation models is complex and requires a lot of expertise. This thesis implements a less complex and effective interaction between a legacy simulation model and RTI using a middleware tool known as Distributed Manufacturing Simulation (DMS) adapter.

Shuttle Model, an Arena based discrete-event simulation model for shuttle operations, is connected to the RTI using the DMS adapter. The adapter provides a set of functions that are to be incorporated within the Shuttle Model, in a procedural manner, in order to connect to RTI. This thesis presents the procedure when the Shuttle Model connects to the RTI, to communicate with the Scrub Model for approval of its shuttle's launch.
ACKNOWLEDGMENTS

I offer my sincere and deepest thanks to my advisor Dr. Luis Rabelo for his continuous motivation and inspiration throughout this work. Dr. Luis Rabelo not only helped me as thesis advisor but also guided me to take important decisions in the academics. I take this opportunity to thank Dr. Jose A. Sepulveda for all the coaching I received from him during the coursework. I also thank Dr. Peter Kincaid and Dr. Bala Jaganathan for agreeing to be on my thesis committee.
# TABLE OF CONTENTS

LIST OF FIGURES........................................................................................................vii

LIST OF ACRONYMS/ABBREVIATIONS....................................................................ix

CHAPTER 1: INTRODUCTION.........................................................................................1

1.1 Preface..................................................................................................................1

1.2 Distributed Simulation..........................................................................................1

1.3 Architecture for Distributed Simulations............................................................2

1.4 Adapter for the HLA............................................................................................3

1.5 Virtual Test Bed....................................................................................................4

1.6 The Space Shuttle Model.....................................................................................4

1.7 Contributions of the Thesis..................................................................................5

1.8 Outline of the Thesis............................................................................................6

CHAPTER 2: HLA, RTI, AND LITERATURE REVIEW...............................................7

2.1 Preface..................................................................................................................7

2.2 High Level Architecture.......................................................................................7

2.2.1 HLA Rules.......................................................................................................9

2.2.2 Interface Specification.....................................................................................11

2.2.2.1 Run Time Infrastructure...........................................................................11

2.2.2.2 RTI Services............................................................................................13

2.2.3 Object Model Template..................................................................................18

2.2.4 HLA in Space Operations..............................................................................20

2.3 Literature Review.................................................................................................22

2.3.1 Alternatives for HLA.....................................................................................22
CHAPTER 5: CONCLUSIONS AND FUTURE RESEARCH…………………………..66

5.1 Conclusions…………………………………………………………………………..66

5.2 Contributions………………………………………………………………………..67

5.3 Future Work………………………………………………………………………….67

REFERENCES………………………………………………………………………….69
LIST OF FIGURES

Figure 1: Virtual Test Bed Environment ................................................................. 4
Figure 2: Conceptual flow diagram for Space Shuttle processing ......................... 5
Figure 3: Functional View of HLA Federation ....................................................... 9
Figure 4: RTI, Federates Communication ............................................................. 11
Figure 5: RTI Components ................................................................................. 12
Figure 6: RTI and Federate Ambassadors ........................................................... 13
Figure 7: Federation Life Cycle ............................................................................ 14
Figure 8: Federate – Federation Interplay ............................................................ 18
Figure 9: Life Cycle of an Object ......................................................................... 19
Figure 10: Federate view of the Distributed simulation ........................................ 21
Figure 11: OMG Reference Model Architecture ............................................... 23
Figure 12: Connections made for RMI ................................................................. 25
Figure 13: WebHLA with CORBA, JAVA, COM domains ............................... 26
Figure 14: Connecting Simulation Tools to RTI using Wrapper classes .............. 27
Figure 15: Integrating HLA and AnyLogic ......................................................... 29
Figure 16: HLA Architecture using Java Adapter ............................................... 30
Figure 17: RTI – GUI communication ................................................................. 30
Figure 18: FMI Structure ..................................................................................... 32
Figure 19: Legacy Simulation Integration Using DMS Adapter ......................... 36
Figure 20: Adapter Methods ............................................................................. 39
Figure 21: Adapter Execution States ................................................................. 40
Figure 22: Working of DMS Adapter .................................................................45
Figure 23: Space Shuttle Hardware Flow .........................................................48
Figure 24: Shuttle Process Flow .....................................................................49
Figure 25: Shuttle Process Flow, HLA, Scrub Model .......................................50
Figure 26: Time Management Block in Arena ...............................................52
Figure 27: Time Management Pseudo Code ..................................................52
Figure 28: Pseudo code to initialize the adapter ............................................54
Figure 29: Input file data for initializing the adapter ......................................54
Figure 30: New scrub logic in Shuttle Model .................................................55
Figure 31: Pseudo Code for constructing a message with shuttle information ....56
Figure 32: Pseudo Code for creating an object ..............................................56
Figure 33: Pseudo Code for receiving and decoding a message ....................57
Figure 34: Pseudo Code for terminating the adapter .....................................58
Figure 35: Screenshot of RTIExec.................................................................59
Figure 36: Run setup of the Space Shuttle Model ..........................................60
Figure 37: Federates are Connected to the RTI .............................................61
Figure 38: A message is received by the Scrub Model ....................................62
Figure 39: A message is received by the Shuttle Model for launch approval ...63
Figure 40: Federation shutting down ............................................................64
Figure 41: Screenshot resultant SIMAN report file ......................................65
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLA</td>
<td>High Level Architecture</td>
</tr>
<tr>
<td>DMSO</td>
<td>Defense Modeling and Simulation Office</td>
</tr>
<tr>
<td>RTI</td>
<td>Run Time Infrastructure</td>
</tr>
<tr>
<td>DMS</td>
<td>Distributed Manufacturing Simulation</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of standards and Technology</td>
</tr>
<tr>
<td>IMS</td>
<td>International Manufacturing Systems</td>
</tr>
<tr>
<td>VTB</td>
<td>Virtual Test Bed</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>FOM</td>
<td>Federation Object Model</td>
</tr>
<tr>
<td>SOM</td>
<td>Simulation Object Model</td>
</tr>
<tr>
<td>OMT</td>
<td>Object Model Template</td>
</tr>
<tr>
<td>MOM</td>
<td>Management Object Model</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>ORB</td>
<td>Object Request Broker</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off-the-shelf</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
<tr>
<td>FDK</td>
<td>Federated simulations Development Kit</td>
</tr>
<tr>
<td>API</td>
<td>Application Programmer's Interface</td>
</tr>
<tr>
<td>KSC</td>
<td>Kennedy Space Center</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
CHAPTER 1: INTRODUCTION

1.1 Preface

A less complex and cost-effective distributed simulations can be formed if reusability and interoperability are added to traditional simulation models. This thesis focuses on providing these properties to the Arena Shuttle Model to join and communicate with other simulation models in the distributed simulation environment, and study its behavior.

This chapter familiarizes the approach of this thesis, introducing the concepts of a distributed simulation and briefly describing the overview of the Shuttle Model, and background of this study. Then this chapter also explains the contributions of this thesis followed by the outline of this thesis.

1.2 Distributed Simulation

A model is a logical description of the behavior of a system, process, or component. Simulation, by definition, is to develop a model, a software representation of the behavior of the system that helps to perform experiments on the model to determine the performance and to predict the effect of changes to the system as time progresses. Simulations mainly are of two types (1) Continuous simulations, where the changes in the state of the modeled system are directly depended on changes in time and (2) Discrete event simulations where discrete entities change the state as events occur in the simulation. The state of the model changes only when the events occur. Many simulation tools and packages are now available to construct an effective model for any real world model. But the stand alone simulation models may grow complex over the time, resulting
in higher cost at maintenance and development. These complexities can be minimized by
developing concurrent environment where some components of the simulation are
executed in parallel over a network. Also, these models can be reused to operate with
other components of the major model saving time and cost for which distributed
simulations would serve the purpose. A distributed simulation is the set of simulations
interoperating asynchronously, in parallel, over a set of processors in a network.

1.3 Architecture for Distributed Simulations

This section introduces the concepts of High Level Architecture [23][24]. Simulations participating in a distributed simulation can be in different packages and
environments, modeled for different purposes. An approach is necessary to make these
models interoperate. A generalized standard is necessary for the simulations to
communicate with each other. Such necessities are overcome by an architecture called as
High Level Architecture (HLA). The Defense Modeling and Simulation Office (DMSO)
developed the High Level Architecture to address its continuing need for interoperability
among its new and existing simulation models.

The development of HLA supports that no single simulation can satisfy all the
uses. A set of simulations interacting under the concept of HLA is called as Federation
where the simulations are Federates. HLA provides the structure that supports the reuse
of capabilities of different simulations for a new purpose. The IEEE defines computer
simulations as "Major functional elements, interfaces, and design rules, pertaining as
feasible to all simulation applications, and providing a common framework within which
specific system architectures can be defined" [23].
The HLA is a set of rules, interfaces and data specification tools which, does not specify any implementation or mandatory usage of the software. RTI is the executable software component of HLA and provides services for federates to communicate and coordinate within a federation. A complete overview of the HLA and RTI is covered in chapter 2.

1.4 Adapter for the HLA

This section introduces the DMS adapter [4]. The implementation of HLA for a federation may be complex and require time and effort. The coding may become more difficult when some simulations do not support the architecture and the same code may need to be developed for each federate to participate in the federation. Since the HLA does not specify a particular implementation, simulations in a federation may follow different implementations to HLA.

An adapter is a software component that works as a middleware between the simulations and the high level architecture, aiming to minimize the problems stated above. National Institute of Standards and Technology (NIST) developed such an adapter called Distributed Manufacturing Simulation (DMS) Adapter for their MISSION project, a part of the International Manufacturing Systems (IMS) Program (Refer www.ims.org). DMS adapter supports integration of various manufacturing simulations with each other and other applications. The DMS adapter can integrate non-HLA compliant simulation tools and non-simulation oriented applications. One of the goals of the adapter is to minimize the changes in the simulation to participate in a federation. An outline of the adapter is discussed in chapter 3.
1.5 Virtual Test Bed

Virtual Test Bed is a computing environment providing the integration of simulation models developed for specific space operations, into a single integrated simulation network [22]. Figure 1 depicts the overview of the VTB. It contains the existing models that NASA KSC has.

![Virtual Test Bed Environment](image)

**Figure 1:** Virtual Test Bed Environment [22]

1.6 The Space Shuttle Model

This section introduces the Space Shuttle Model [13][14]. The Shuttle Model is the simulation model for the Kennedy Space Center (KSC) shuttle operations. The Shuttle Model represents a probabilistic simulation model of the operational lifecycle of the Space Shuttle flight hardware elements processing through their respective ground facilities. The model includes the shuttle’s major flight hardware elements, i.e. orbiters, external tanks, Space Shuttle main engines, and solid rocket motor/booster elements.
processing through their respective facilities along with the major supporting elements of ground support equipment.

Figure 2: Conceptual flow diagram for Space Shuttle processing [29]

The Shuttle Model also covers flight operations such as ascent, mission duration, and landing as the processing of the next flow depends on the flights return to the KSC.

Arena, a discrete-event simulation development tool, is used in this shuttle simulation.

1.7 Contributions of the thesis

The Shuttle Model is the legacy simulation model in this thesis. The Shuttle Model is the integral part of the Virtual Test Bed (VTB) simulation network of NASA for the space operations. There are many methodologies for developing a HLA compliant
Shuttle Model to be part of VTB. The thesis implements DMS Adapter to use as middleware for the High Level Architecture and the Shuttle Model. The Shuttle Model is configured for the DMS adapter and is connected to the HLA/RTI.

Visual Basic Applications in Arena makes use of the libraries provided by the NIST for the DMS adapter, to control the simulation, join the federation and communicate with other federates. Complete development and testing is achieved through many steps discussed further in this thesis.

1.8 Outline of the Thesis

Chapter 1 introduces the concepts used in this thesis. The contributions section outlines the problem to be solved. Chapter 2 overviews the High Level Architecture followed by the literature review that outlines the various existing methodologies for solving the problem. Chapter 3 deals with the DMS adapter, its overview and usage in space operations simulation models. Chapter 4 is where the development process is discussed step by step followed by testing the model and analyzing the results. Chapter 5 presents the conclusions, scope and ideas for future work.
CHAPTER 2: HLA, RTI, AND LITERATURE REVIEW

2.1 Preface

This chapter presents an overview of High Level Architecture, its components and working. Some existing alternatives of HLA are then discussed later in the literature review. Discrete-event simulation packages like MODSIM III and AnyLogic are presented. Then some existing adapters for integration of COTS simulation languages and HLA are presented.

2.2 High Level Architecture

The dramatically increased bandwidths and processing capabilities of the future high speed computer networks make possible many real time distributed systems. Monolithic simulations can be connected through the networks to form a single effective simulation. The purpose is to reduce the time and cost for building a single simulation from scratch while, some existing simulations can be reused as the components for the new simulation. Also the execution time of the new simulation is drastically reduced as parallel computing is possible though many processors over the network. Individual simulations can be run in parallel. To achieve this, it requires a framework to make the simulations reusable and able to communicate with other simulations across the network. Also these simulations may have been developed using different simulation development packages and tools over different operating systems. The requirement is immediately recognized by the Department of Defense (DoD) to make the best use of their existing and future simulations. Under the leadership of Defense Modeling and Simulation Office
(DMSO), to support reuse and interoperability across the large number of different types of simulation, High Level Architecture (HLA) was developed. HLA is based on the idea that no single simulation can satisfy all uses and users.

HLA is object based language independent software architecture, specifying rules and information exchange policies across the distributed simulation network. Using the HLA framework individual simulations can join a network of simulations forming a distributed simulation network which in the terms of HLA called as the Federation. The individual simulations that form the federation are called as the Federates. HLA separates the functionality for the individual simulations from the infrastructure required for interoperability among simulations [23].

Overview

HLA is a constitution of three components. First are the simulations themselves. There is no constraints for the simulations to become federates. A federate can be an individual computer simulation, a code developed in any language which has nothing to do with simulation, or even an interface to monitor the processes. The only requirement is that federates should be capable of exchanging objects as the date transfer in HLA is thorough the objects supported by the services provided by the Run Time Infrastructure (RTI). The second of the components is the RTI that provides a set of services for the interaction among federates, and management support functions.

Runtime interface, that provides a standard way to the federates to interact with the RTI, is the third component of HLA. The runtime interface specification aids the federates to invoke the services provided by the RTI to support runtime interactions
among federates, and to respond to the requests by RTI. This implantation is language
independent.

![Functional View of HLA Federation](image)

Figure 3: Functional View of HLA Federation [24]

HLA also supports the passive collection of data and the monitoring of simulation
activities.

2.2.1 HLA Rules

This section covers the rules for HLA [41]. The rules that HLA demands, ensure
proper data transfer though objects among the federates and interactions between the
federation and its federates. They define the responsibilities of each simulation to
participate in a federation.

**Federation Rules:**

1. Federations shall have an HLA Federation Object Model (FOM), documented in
   accordance with the HLA Object Model Template (OMT).
2. In a federation, all representation of objects in the FOM shall be in the federates, not in the run-time infrastructure (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 run-time infrastructure (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 an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template (OMT).

2. Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM.

3. Federates shall be able to transfer and/or accept ownership of attribute dynamically during a federation execution, as specified in their SOM.

4. Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of 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 exchange with other members of a federation.
2.2.2 Interface Specification

Interface specification is the standard provided by the HLA for the federates to interact with the RTI. This section first explains RTI followed by the six basic RTI service groups [23][24].

2.2.2.1 Run Time Infrastructure

Run Time Infrastructure (RTI) is software that implements the HLA interface specification [18]. It provides a common service to all federates in a simulation. RTI separates the functionality and the necessary communication of the simulation.

![RTI, Federates Communication](image)

Figure 4: RTI, Federates Communication [18]

All information exchanged during a federation execution is passed through the RTI. The RTI is composed of three components, the RTI executive process, the federation executive process and the libRTI library.
Federation Executive Process (FedExec)

Federation executive process manages the joining resigning, and the data exchange processes among the joined federates. RTI creates a FedExec process when the federation is created and a federation joins it. It is eventually destroyed before the completion of the federation execution, when all the federates leave the federation.

RTI Executive Process (RTIExec)

The purpose of the RTI executive process is to manage all the federation executions. Each FedExec is assigned a unique identification. It ensures the federates join the appropriate federation. There may be any number of federations under one RTIExec. Even then, the communication between federations is not possible.

libRTI Library

HLA specifies that data to be exchanged between the federates should pass through the federates, and it is accomplished by the two classes of libRTI namely RTI ambassador
and Federation ambassador. libRTI enables the access for the federates to use the services provided by these classes.

RTIAmbassador has the services to pass the information between the RTI and the federate. It should be incorporated into each federate executable. FederationAmbassador is an abstract class, and each federate should provide an implementation of its services to join a federation.

2.2.2.2 RTI Services

As stated, the RTI's primary function is that of a data distribution mechanism. Federates send information through the RTI which distributes the information to the appropriate parties. The RTI does not maintain information about the state of the federation. It does not handle any semantics associated with the interaction between the federates. The RTI does not specify the exact byte layout of data sent across the network. RTI provides a common set of services to the federates. They are divided into six categories [18].
**Federation Management**

The Federation Management services provide operations for federation execution like creating, destroying federations, federate joining and resigning from federations, and federation wide operations like controlling synchronization points, support state saves and restores. Creating the federations is done by the RTIAmbassador class. The function created by the Federationexecution() requests the RTIExec process to create a federation. If no federation exists with the supplied name, the RTIExec process continues to create the federation and assign it with the given name. If the federation already exists, an exception is caught warning that a federation with the supplied name is already active and the federates are joined to the existing federation. JoinFederationExecution() is the function that is to be called if the federates want to join a federation.
ResignFederationExecution() is called by a federate to leave from the federation. DestroyFederationExecution() tends to terminate the federation.

**Declaration Management**

Declaration Management provide services for the federates to generate information. Invoking the declaration management services, a federate can register object instances, update instance attribute values, and send interactions. Federates should explicitly declare what they are able to publish, to produce objects or interactions. Similarly, the federates requesting an object or interaction should publish what they seek in the attribute. Once published, the information is available throughout the federation. All the classes and attributes used to declare must be consistent with the Federation Object Model (FOM). Based on the interests of the federates, the RTI controls the distribution of the information. A federate generates objects or interactions only if at least one federate is interested in the information published by it. A federate should declare to stop publishing or subscribing if it is no longer interested in the object or interaction declared.

**Object Management**

Object Management services with the registration, modification, and deletion of object instances, and send/receive interactions. Object management includes instance registration and instance updates on the object production side and instance discovery and reflection on the object consumer side. Objects must be published using the declaration management services before creating or subscribing. All the subscribing federates must be notified before deleting an object owned by a federate. Object management services support the exchange of information. Its methods are associated with sending and
receiving interactions, and controlling instance updates. A Reflect of a value update is requested by a federate subscribing the object.

Ownership Management

Ownership management is used by federates and RTI to transfer ownership of instance attributes among federates. It gives a responsibility to federates to update attribute values of an object instance. A federate must have the ownership of the object instance to update its attributes. Ownership can be transferred among the federates through RTI. Only one attribute can have an ownership over an object attribute at a given time so as it can only be the federate to update the attributes or delete the object. Ownership can be transferred if the federate who possess the object or the federate who requests the object, initiates it. Only the federate with a privilege to delete can delete the object. To avoid further publishing, RTI informs all the owners of the object’s attributes that the object is deleted and no longer exists. It is also possible that one federate can own all the object attributes at a given time.

Time Management

The federation time axis maintains points as representation of time in the system. Federates advance along this axis throughout the execution of the federation. Each federation time advances may be depended on the time advances of other federates in the system. Time management services are concerned with providing the mechanisms for controlling the federate advancement along the federation time axis. To maintain the order of the information delivered to the federates, time management services coordinate with the object management services. Federates associates some of their activities to the points on the federation time axis by assigning time stamps to the activities. The activities
of federates can be either regulating or time constrained. It is the responsibility of the federate developer to use proper time stamp mechanism to fit the purpose of the federate in the federation.

**Data Distribution Management**

Federates use Data Distribution Management (DDM) to reduce the transmission of irrelevant data as well as reception of it. Data distribution mechanisms refine the data at the attribute instance level unlike declaration management services provide information on the data at class attribute level. Federates producing the data, employs DDM services to define the properties of data in terms of user defined spaces. Federates requesting the data, specify their requirements in terms of same spaces using DDM services. RTI matches the properties of the data defined by the federates with requirements of data also defined by federates, to distribute the data effectively.

An overview of the services provided by the RTI in a federate life cycle is given in the Figure 8. The figure explains the role of RTI and its services in aiding a federate interact with the Federation.
2.2.3 Object Model Template

The HLA Object Model Template (OMT) provides a template for documenting HLA-relevant information about classes of simulation or federation objects and their attributes and interactions. This common template facilitates understanding and comparisons of different simulations and federations, and provides the format for a contract between members of a federation on the types of objects and interactions that will be supported across its multiple interoperating simulations. Figure 9 outlines the lifecycle of an object of a federation [19].
The purpose of an HLA Federation Object Model (FOM) is to provide a specification for data exchange among federates in a common, standardized format. The content of this data includes an enumeration of all object and interaction classes pertinent to the federation, along with a specification of the attributes or parameters that characterize these classes. The individual components of an HLA FOM establish the “information model contract” that is necessary to achieve interoperability among the federates.

Simulation Object Models

An HLA Simulation Object Model (SOM) is a specification of the intrinsic capabilities that an individual simulation could provide to HLA federations. The standard
format in which SOMs are expressed facilitates determination of the suitability of simulation systems for participation in a federation.

**Management Object Models**

Management Object Models (MOM) defines a set of objects and interactions to manage a federation. Defined as a standard part of FOM in the interface specification, it can be extended by adding attributes or subclasses of an object, or by adding interactions into the MOM. The state of the federates and the federation like adjust federation, request information etc are manipulation by the interactions defined in the MOM. RTI creates and manages the object instances defined in the MOM and updates its attributes.

**2.2.4 HLA in Space Operations**

This section provides an illustration of HLA in space operations, based on the models existing in the VTB architecture. Considering the part of Shuttle Model, where the Scrub logic is implemented, and the logic is based on a historical data with a random time generation that is close to the historical data and the shuttle is hold in the scrub logic until the random time is delayed. Such probability is based on several factors like unfriendly weather conditions, launch area intrusions etc. A Scrub Model is developed to calculate these factors to maximum accuracy by using real world data. The Shuttle Model, instead of relaying on randomly generated, can get the delay time from the Scrub Model which is more accurate. This is more real-time and efficient as the scrub logic is based on the factors affecting a particular instance of time. HLA can be used to model this scenario.

The Shuttle Model and Scrub Model are both developed in Arena. Arena simulation can be externally controlled by manipulating the model as a SIMAN object in
visual basic editor provided by arena. To make the models as federates, a library in C++ is to be developed first and included in the Visual Basic editor. The C++ library is a set of APIs for the external module libRTI. Both RTIAmbassador and FederateAmbassador are included in the C++ library. An implementation of abstract functions of the FederateAmbassador is to be developed. This library is included as a dynamic link library (dll) in Visual Basic of the arena, where internal operations are handled with the functions provided by Arena software, and SIMAN objects. External operations like join federation, communicate with RTI is handled by the C++ library just developed.

![Diagram of Federate view of the Distributed Simulation](image)

**Figure 10: Federate view of the Distributed Simulation**

The Shuttle Model can then publish objects that have the attributes of launch time, shuttle ID etc and can also publish objects that has attributes like launch approval.
Similarly the Scrub Model can publish the objects it wants to request/create. As the publishing is done, both the models get an approval to create and update objects as there is at least one federate that is interested in them i.e. each federate requested the objects the other want to create. The Shuttle Model can start its execution by creating and joining the federation and update the object created with the shuttle information when the shuttle is ready to launch. RTI reflects the change through out the federation. The updated object is reflected in the Scrub Model which in turn calculates the delay based on the information provided and updates the shuttle launch time which is then reflected in the Shuttle Model object. This coordination continues until the Shuttle Model finishes its run and finally resigning from federation. The Scrub Model also resigns and the federation is destroyed.

HLA can further be used in integrating other models of the VTB. It provides a collaborative environment for virtual spaceport engineering.

2.3 Literature Review

The Literature review is divided into three sections. The first section focuses on the architectures to build a distributed simulation environment. The next section is a research on integration of discrete-event simulation tools and HLA. The final section gives an overview of some existing adapters that can be used for the Space Shuttle Model to integrate with HLA.

2.3.1 Alternatives for HLA

This section provides overview of some existing architectures that can be used in the place of HLA.
CORBA

This section introduces to the CORBA architecture. The Common Object Request Broker Architecture (CORBA) is an emerging open distributed object computing infrastructure being standardized by the Object Management Group (OMG). CORBA automates many common network programming tasks such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching. CORBA is an older architecture than HLA with a large set of specifications and protocols [2].

![Figure 11: OMG Reference Model Architecture](image)

The above figure overviews the architecture of CORBA. All the interfaces are developed in a specified language called as Interface Definition Language (IDL). Common facilities are interfaces for end-user applications where as application and domain interfaces go by their name. Object services are domain-independent interfaces that are used by many distributed object programs. Such services can be a naming service that allows clients to find objects based on names, a trading service that allows clients to
find objects based on their properties, or any other service that fulfills its role. Object Request Broker is the object manager in CORBA.

The Object Request Broker (ORB) is the middleware that handles the communication details between the objects. Any local or remote objects are enabled to receive or send messages. All messages between client and server must go through the ORB. The client makes requests of a specific server on a specific machine. The clients only have to know how to ask an ORB for services, and the servers only have to register their services with the ORB.

ORB is software that resides in both client and server machines. When a client makes a request, it first goes to the ORB in the client machine. The client ORB makes a connection with the server ORB. Similarly, when the server answers the request, it sends it to the server ORB and the client ORB receives from there. The server can either statically or dynamically provide remote objects. This involves stubs – a link of an object to the ORB on its machine and generated by IDL, and skeletons – the server stub. The client stub and the server skeleton can be in any language independent of each other.

**RMI**

Remote Method Invocation (RMI) is a part of the standard java libraries, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. RMI is a mechanism that allows one to invoke a method on an object that exists in another address space. The RMI mechanism is basically an object-oriented RPC mechanism. There are three processes that are involved in supporting the RMI. Client – that invokes a method on remote object, Server – that owns the remote
object, and Object Registry – server that relates objects with names. RMI uses JAVA as its object interface language [2].

All the remote interfaces in the client side and the server side are compiled separately. In order to server the remote objects, the objects may be a subclass of java.rmi.server.UnicastRemoteObject. Remote objects subclass implements UnicastRemoteObject to define desired interface. The remote object is instantiated and bound to a name using java.rmi.Naming class. As the remote objects are involved in potential security risk, RMI requires an instance of java.rmi.RMISecurityManager to implement a security policy. RMISecurityManager determines if the invoked methods are local or remote, protecting against unsafe operations.

![Figure 12: Connections made when RMI is used](image)

The methods of java.rmi.registry.Registry bind objects with names, listing all available objects on a server. The client uses this registry to invoke appropriate methods. JDK’s inbuilt networking capabilities are used by RMI for remote constructions. Non Java applications can be interacted by RMI using Java Native Interface (JNI).
WebHLA

This section introduces to the concepts of webHLA [10]. WebHLA is an extension to HLA. HLA is joining with emergent standards of web based distributing computing like – CORBA, Java, COM and W3C COM. It is developed to offer a powerful modeling and simulation framework, capable to address new challenges of DoD computing in the areas of simulation based Design, Testing, Evaluation and Acquisition.

WebHLA is developed over a 3-tier architecture. Java Web Object Request Broker (JWORB) is multi-protocol server capable to manage objects of various distributed object models and CORBA, Java, COM and XML. HLA is supported using the Object Web RTI (OWRTI), an implementation of RTI 1.3 based on Java CORBA.

Figure 13: WebHLA with CORBA, JAVA, COM domains [10]

WebHLA manages database tools like event logger, event database manager, event playback federate to save entire simulation segments for later analysis.

WebHLA is a future version of HLA. RMI, CORBA and HLA take almost similar approaches for distributed computing. But simulations involving legacy simulation models, HLA provides better functionality for its simulation related services.
2.3.2 HLA Interfaces of Simulation Tools

A general approach for the simulation tools to communicate with Runtime Infrastructure (RTI) can be done using wrapper C/C++ libraries. The wrapper libraries translate the RTI functionality to the simulation tool. As described earlier, RTI communicates with a federate using the federate’s FederateAmbassador and federate communicates with the RTI using the RTI’s RTIAmbassador. These ambassadors can be termed as objects where the communication is done by invoking the methods of these objects. By writing the code for these objects in the wrapper libraries and finding a way for binding the wrapper libraries and the simulation tool, any simulation tool can be developed to be a federate in a HLA federation.

Figure 14: Connecting Simulation Tools to RTI using Wrapper classes [38]

Figure 14 represents the role of the wrapper libraries. Time management depends on the internal representation of the logical time in the simulation tool.

Glen D. Johnson [12] proposes the networked simulation applications using HLA with MODSIM III as reference simulation tool. MODSIM III is a Commercial off-the-shelf (COTS) object oriented simulation language developed by CACI. MODSIM III is
built to bind to the HLA standards. Its interface is integrated with other parts of MODSIM system such as SimGraphics, to provide a level above the HLA to better support message pumping, exceptions and callbacks. MODSIM III with HLA support provides universal data representation. With this capability, data can be transferred “on the wire” between two computers and operating environments. This discrete-event simulation language can save a significant time and risk in building HLA compliant simulation model federates.

Borshchev et al. [1], describes AnyLogic simulation tool and its integration with HLA for a distributed simulation environment. AnyLogic is a general purpose simulation tool for discrete, continuous, hybrid simulations. It is developed by XJ Technologies. AnyLogic has a graphical model editor for developing simulations. It comes with a code generator that converts the model into Java code. AnyLogic is windows-based, with data collection and analysis facilities, debugging and visualization tools. XJ Technologies developed HLA Support Module as an add-on package for the AnyLogic and RTI interaction.
HSM uses the stepHook interface to enable AnyLogic models to exchange messages and synchronize Local Simulation Time to federation Global Time with the federation. AnyLogic is an environment for programming in Java with simulation class library for visualization specification support. This property makes it easy to develop HLA based extensions for a model to participate in a distributed simulation.

2.3.3 Adapters for COTS simulation languages

Java Adapter for HLA/RTI

The Java Adapter for HLA/RTI was developed by University of Ottawa, Canada to implement a project called Agent Aided Collaborative Virtual Environments over HLA/RTI. Classes and Functions developed in Java constitute the RTI Library. Java invocation calls are used to invoke the services of HLA. Object management is handled by the Java object controller [44].
The above figure describes the interaction between the local objects and the RTI. RTI interface is a set of Java applets aiding the user monitor and interact with the existing system. A simple communication between two local scripts communication through RTI is

Where the steps are,

1. A user selection
2. This causes a local script to run.
3. This in turn invokes the callback event by EAI in the Java applet, which then retrieves the event asks the RTI to send attribute update through JNI.
4. RTI passes the event and invokes the remote federate ambassador to reflect updated attribute. The latter one is “RTI initiated” service.
5. The Java applet retrieves the message to its browser.
6. The browser then converts the message to an event.
7. The event causes execution of the local script.

**CORBA IDL Middleware for HLA**

CORBA is an acronym for the Common Object Request Broker Architecture. It is an object-oriented, language-independent architecture that is used to share an object’s data and methods over a network. The shareable portion of these objects is specified in an Interface Definition Language (IDL) that serves an interface specification function similar to the headers files common in C [36].

The IDL4HLA compiler outputs CORBA-like stub and skeleton source code files. Object implementations, server, and client applications access each object through its corresponding stub or skeleton. Although these stubs and skeletons look identical to their CORBA equivalents they are actually performing HLA operations behind the scenes.

Common HLA operations are encapsulated in a Translation Library that serves the same function as the CORBA ORB’s network libraries uses the IDL4HLA compiler also outputs OMT files. CORBA attributes are mapped to HLA objects and CORBA methods are mapped to HLA interactions. The combination of this information with the Translation Library provides all of the functionality typically needed to interact with the RTI's library of communication routines. Interfaces to both the Translation Library and/or
the RTI's library itself are exposed so that developers can tune their systems for atypical situations.

**PRTI 1516**

Pitch technologies developed this high level RTI adapter to facilitate the communication between federates using different versions of HLA (1.3 and 1516). For this the adapter is divided into two adapters – PRTI adapter and 1516 adapter. Legacy Simulations mould on 1516 adapter to establish connection with the RTI. Java and C++ APIs are used to build the adapter and the interfaces [33].

**Federated simulations Development Kit (FDK)**

![FMI Structure](image)

Figure 18: FMI Structure [9]

Federated simulation Development Kit (FDK) is the library of software modules developed at Georgia Institute of Technology, by Parallel and Distributed Simulation (PADS) research group [9]. This library offers the modules for building run-time infrastructure (RTI). RTI developers can choose from the set of provided FDK modules for developing RTI implementation. RTI developers can benefit from incorporating these available modules, instead of developing on their own. FDK is another example of how
predefined simulation modules can help developer to build the model. The adapter aims at promoting research in designing and building new, faster and better RTIs.

2.4 Synopsis

The Literature review presents various existing packages for developing simulations. It also reviews some architecture to develop a distributed simulation network. Arena is the simulation package used to develop the simulations in this thesis and High Level Architecture is used. Since Arena is not a HLA-Compliant, DMS adapter is used to connect it to the RTI. A detailed vision of the adapter is presented in the next chapter.
CHAPTER 3: DISTRIBUTED MANUFACTURING SIMULATION

ADAPTER

3.1 Preface

Distributed Manufacturing Simulation (DMS) adapter is developed by National Institute of Standards and Technology (NIST) for their MISSION project. DMS adapter is used to accomplish the integration between Space Shuttle Model and HLA [4].

This chapter describes the architecture of the DMS adapter for RTI. It briefly discusses the necessity of the software adapter followed by its architecture, methods and properties.

3.2 Need for a software adapter

The goals of a distributed simulation system can be summarized as to provide multiple levels of simulation systems across multiple organizations where any information from each organization can be hidden, with low cost runtime models. The aim is also to achieve what a single simulation model cannot. The HLA-RTI model is supposed to function with a less cost and less complexity. But the following issues may affect the usage of HLA-RTI model [4].

- There are 120 service calls in the RTI Interface. RTI uses the implicit-invocation approach to each call. The operation of each call depends on the current state of the RTI. Because of the implicit-invocation approach, it may not be possible to check if a call is succeeded. It may also require timeouts and rollbacks.

- Since many COTS discrete-event simulation package do not support this architecture, integrating with HLA may result in very complex coding and effort.
• Federation development process is very complex and requires a lot of study and expertise.
• A federate can use any time management policies like Time Stamp Ordered (TSO), Receive Ordered (FIFO). Management of a group of federates using different time management policies increases complexity.
• HLA is designed to handle small items. A distributed simulation may have thousands of objects but, each object can only be represented as integer. It requires a significant effort to manage anything other than integers, floats or strings.
• HLA tools, services, expertise, and software are expensive to acquire.
• Simulations created with one RTI may not work with another RTI.

Hence, developing the Federate Ambassador and adaptation code can be a significant effort when developing a distributed simulation, and that this effort must be repeated for each legacy simulation that is to be integrated.

3.3 DMS Adapter

Distributed Manufacturing Simulation Adapter (DMS) can be used to minimize the effort. Instead of having legacy simulations integrated directly with the HLA/RTI, those simulations will interact with the interface of the adapter. The goal of the adapter is to provide a simplified method for integrating legacy simulations into distributed simulations while also providing as much of the capabilities of the HLA/RTI as possible. Adaptation code is to be developed to integrate a legacy simulation system with the DMS Adapter. However, by reducing the complexity of the interface to which the legacy
simulation is being integrated, the level of effort for performing the integration is greatly reduced

3.4 Goals of the DMS adapter

- Simplify the process necessary for distributed simulations using existing tools.
- Maintain storage for remote and local object instances
- Provide the capability to use the same adapter in multiple federations.
- Support Adapter parameterization with externally defined data
- Perform housekeeping tasks without disturbing the simulation associated with the adapter
- Work cooperatively with the Federation Manager to manage time advancement and synchronization

3.5 DMS Adapter Architectural Goals

This section defines the architectural goals of the DMS adapter [4]

- Minimize complexity of the interface
The interface of the adapter has only 35 methods compared to the 120 methods with 40 callbacks defined by the RTI and Federate Ambassadors.

- **Take care of Federate Ambassador implementation issues**

As stated in previous chapters, a simulation must implement the FederateAmbassador to join the federation. But with the DMS adapter legacy simulations will not have to develop Federate Ambassador implementations. The adapter will implement a federate ambassador and use it to receive information from the RTI.

- **Define an interface that integrates with procedure oriented legacy simulation models.**

After the execution of the adapter’s interface methods, the results are returned immediately to the legacy simulation. A message queue is maintained in the federate’s adapter to store the information that is passed back asynchronously to a federate. This includes information that is generated by the activities of other federates in the federation. The adapter stores this information and provides methods to access this information upon request from the legacy simulation.

- **Generic FOM objects will contain XML strings to minimize the impact of changes to the information model.**

Developing a different FOM for each distributed simulation configuration is not necessary as the information about the classes and attributes for the objects that are to be exchanged will not be defined in the FOM. A generic object is defined in the FOM and this object is exchanged between federates. This generic FOM object contains an XML string that contains the semantic content for the object. The XML string is the information that will be passed to the legacy simulation. The generic FOM object also
contains information about the type of data contained in the XML string. This facilitates filtering and routing of object updates by the RTI.

- **Maintain storage for the objects that are to be exchanged between simulations**

  The definition of an object that is to be exchanged between simulations differs from the internal definition that each simulation supports for that object. Since each legacy simulation only provides storage for its internal objects and the RTI provides no mechanism for the storage of objects, storage and maintenance for the objects that are to be exchanged is provided. The DMS adapter will provide this capability. Each adapter will provide methods that allow a legacy simulation to create, modify, and delete objects that can be shared with other federates in the federation. Objects will have "owners", and ownership will be granted initially to the adapter (and associated legacy simulation) that created it. Each DMS Adapter uses the services of the RTI to distribute object update information for the objects it owns, and incorporates object update information it receives about objects owned by other DMS Adapters.

- **Simplify time coordination**

  The adapter implements a "time-stepped" synchronization approach. DMS Adapter methods are provided to declare that the associated legacy simulation wishes to advance to a certain simulation time, and to check if it is okay to advance to this time. DMS Adapter indicates to the legacy simulation when to advance, and the legacy simulation can then "simulate" from its current simulation time to the new simulation time that it requested. It can then use the other methods in the DMS Adapter interface to get information about what was going on in the rest of the federation while it was executing
its "simulation step". When all of the simulations use this method, the functionality of the RTI time management services ensures that the collective advancement of all of the simulations proceeds properly.

3.6 Adapter Methods

- Simulation Execution Management Methods

  These methods control the simulation. Ex: INITIALIZE, TERMINATE.

- Object Management Methods

  These methods manage the object life cycle. Ex: CREATEOBJECT, DELETEOBJECT.

- Message Management Methods

  These methods manage the messages exchanged. Ex: SendMessage, GetMessage.

<table>
<thead>
<tr>
<th>Simulation Execution Management Methods</th>
<th>Initialize()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Terminate()</td>
</tr>
<tr>
<td></td>
<td>AdvanceSimulation()</td>
</tr>
<tr>
<td></td>
<td>SimulationAdvanceCompleted()</td>
</tr>
<tr>
<td></td>
<td>GetExecutionState()</td>
</tr>
<tr>
<td></td>
<td>GetSimulationTime()</td>
</tr>
<tr>
<td></td>
<td>GetProperty()</td>
</tr>
<tr>
<td></td>
<td>SetProperty()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message Management Methods</th>
<th>GetNextMessage()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AllMessagesReceived()</td>
</tr>
<tr>
<td></td>
<td>SendMessage()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object Management Methods</th>
<th>CreateObject()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DeleteObject()</td>
</tr>
<tr>
<td></td>
<td>UpdateObject()</td>
</tr>
<tr>
<td></td>
<td>GetObjectValue()</td>
</tr>
<tr>
<td></td>
<td>GetObject()</td>
</tr>
<tr>
<td></td>
<td>SeizeObject()</td>
</tr>
<tr>
<td></td>
<td>ReleaseObject()</td>
</tr>
<tr>
<td></td>
<td>SelectObjects()</td>
</tr>
</tbody>
</table>

Figure 20: Adapter Methods [15]
3.7 Adapter Execution States

The adapter execution states represent the expected lifecycle stages of simulation application, in a complete simulation run [17].

![Diagram of Adapter Execution States]

**PreInitializing**

Adapter is in the preinitializing state once successfully started. Simulation execution methods `Initialize()` and `Terminate()` from this state. `Terminate()` terminates the simulation where as `Initialize()` method causes the adapter to transit to the Initializing state.

**Initializing**

A successful invocation of the `Initialize()` method transits the adapter to Initializing state. Other methods can be called to setup the adapter for the start of a run. Adapter properties
can be set, objects that will be exchanged between adapters can be created, and messages that are to be sent to other simulations at simulation start time can be created, in this state. At this point, the AdvanceSimulation() method is called to indicate that the simulation is ready to start the simulation run.

**ReadyToRun**

This state indicates that initialization of the adapter has been completed, and that the simulation has indicated that it is ready to start the execution at simulation time zero. The adapters in a distributed simulation will synchronize with each other and the federation manages so that all of the simulations start at the same time. While in this state the SimulationAdvanceCompleted() method will return false. When all of the adapters in a distributed simulation have reached this state, they will receive an asynchronous message from the federation that transits them to the Running state. This process is managed by the federation manager and the adapters without the intervention of any of the associated simulation applications. Once an adapter has transitioned to the Running state, the SimulationAdvanceCompleted() method will return true.

**Running**

This state indicates that the distributed simulation run has started. Almost all of the methods are valid to be called in this state, but there may be other pre-conditions that must be also met for successful method invocation. The distributed simulation run will continue until the Terminate method is called, which will transit the adapter to the Terminating state, or until an unrecoverable error occurs, which will transit the adapter to the Failed state.
Failed

When any unrecoverable error occurs in the any of the states, the adapter transits to Failed state. The GetExecutionState(), GetSimulationTime(), and GetProperty() methods may be called to assess the probable reason for failure of the adapter. The only other method that is valid in this state is the Terminate() method, which will attempt to do a controlled shutdown of the adapter and transit to the Terminating state.

Terminating

This state is only a temporary state and it is probably not detectable by the simulation application connected to the adapter. When the Terminate() method is called from another state, the adapter transits to this state, attempts to release all of its resources and terminates its process. Therefore, by the time Terminate() returns to the calling simulation application, the adapter process probably no longer exists. Therefore, no methods should be called once the adapter reaches this state.

3.8 Internal Data Descriptions

The adapter maintains internal repositories of information to carry out its tasks. The adapter makes use of the information in these repositories to implement the methods in its interface. This information is referred to as the adapter’s “internal data” [15].

Federation Member List

This list contains the names for the other simulations applications that are a part of the distributed simulation.

Time Management Data

This data contains the information needed to allow the adapter to work with the RTI in organizing the orderly advancement of time in a distributed simulation.
Local Object Cache
This cache contains the information associated with the objects of the adapter. Objects are called as “local objects.” When an adapter modifies a local object, information about the changes are sent to the other adapters in the federation. Those adapters use this information to keep the remote copies of each object up to date.

Remote Object Cache
This collection contains the information about objects of other adapters in the federation. Objects here are called as “remote objects.” The adapters in a federation work cooperatively to exchange the object update information so that the remote objects can be kept up to date.

Incoming Message Queue
This collection contains the messages that have been sent to an adapter from other adapters in the federation. Methods are provided in the interface to allow a simulation application to receive the messages in the proper time sequence.

Outgoing Message Queue
This collection contains the messages that are to be sent to other adapters in the federation. The messages contained in this collection were created by calling methods in the adapter’s interface. Adapters queue outgoing messages so that those messages can be sent to the other adapters in the proper order and at the appropriate time.

Adapter Properties
This collection contains information in the form of name-value pairs that allow the adapter to be customized for use in a particular distributed simulation run. Adapter methods have been defined to set and get the names and values for adapter properties.
Subscription & Filtering Data

This collection of information defines the set of object and message types that are of interest to an adapter.

3.9 Working with an example

DMS adapter provides a simple integration of simulation tools like Arena with HLA/RTI. Considering the example in the previous chapter where HLA can be used to integrate the Space Shuttle Model and Scrub Model, this example extends the solution using the DMS adapter. Outlining the example, the Space Shuttle Model is a simulation model of the operational life cycle of a Space Shuttle starting from hardware elements processing through ground facilities to return of the shuttle from the orbit. Scrub Model represents a simulation model handling the probabilistic errors before the launch of the shuttle. The model checks for any errors in the shuttle, weather probabilities and delays the shuttle until all the conditions are positive. Instead of using a probability in the Space Shuttle Model for scrub logic, Scrub Model can be used as an efficient alternative. A distributed simulation network can be established between the Space Shuttle Model and the Scrub Model where the Space Shuttle Model asks the Scrub Model for a launch approval before the launch of the shuttle to its orbit. The Scrub Model receives the request and checks for probable errors and delays the flight until the errors are recovered. Then the Scrub Model returns the launch approval to the Space Shuttle Model. Once the approval is received the Shuttle Model continues its launch. Hence, the goal is to manage the communication of the Space Shuttle Model with the Scrub Model using HLA and DMS adapter as the middleware used for integration of HLA and the simulation models.
Both the simulation models used in this example are developed in Arena, a discrete-event simulation language. DMS can be used along with the arena for it to join the federation. Figure 22 shows the architecture.

![Diagram of DMS Adapter](image_url)

**Figure 22: Working of DMS Adapter**

DMS adapter, a set of dynamic link libraries (dlls), is included in the visual basic application of the arena simulation helps the simulation model to connect to the HLA, create a federation, join the federation, create objects, exchange objects among federates and finally terminate federation. Siman object of the arena simulation, in the visual basic applications help manually to control the simulation like, to set value for a variable, to create or route or dispose an entity etc. To manage these utilities, a communication network is developed.

The adapter implements the FederateAmbassodar for the simulation models. All the simulation models need to do is to use the functions provided by the adapter. Working
with the scenario described above, the Space Shuttle Model initializes the adapter at the start of its run. The adapter creates a federation and joins it. The Scrub Model also initializes its own adapter at the start of its run. As the federation is already created, it fails to create a new federation, and joins the existing one. Both the federations are in initialize state and pop up a message box that they are in a federation. Accepting the messages, both the models are set to running state. Each adapter comes with a message queue of its own. The queue is to check for messages at regular time intervals. The Space Shuttle Model starts to run and the shuttle after the hardware elements processing, is ready to launch. A request is made at this point to the Scrub Model to check for errors. An object is created with “request for launch”. The message is sent to the adapter of the Scrub Model. The message is added to the message queue of the Scrub Model. As stated, the Scrub Model checks for messages regularly. Once it finds out a message in its queue from the Shuttle Model, it starts its run and holds the shuttle for all the possible errors. After a successful run, it sends back the “launch approval” message to the Shuttle Model adapter’s message queue. As the message is interpreted by the Shuttle Model, the launch of the shuttle is continued.

The technical details of this example and its detailed working are described in the next chapter where the Space Shuttle Model is implemented for HLA integration and tested.
CHAPTER 4: IMPLEMENTATION OF THE CONNECTION
BETWEEN LEGACY MODEL AND RTI

4.1 Preface

The Space Shuttle Model is a simulation model for the operational life cycle of Space Shuttle flight hardware elements processing through their respective ground facilities and the orbit of the shuttle and return of the shuttle. Such model is a monolithic simulation model with pre-recorded probabilistic values used when needed. These values are depended on the history of records maintained. There are simulations existing that generate these probabilistic values with a maximum possible accuracy. Such simulations along with Space Shuttle Model, if in a communication network, will provide a huge simulation network with a great increase in efficiency compared to the solo Space Shuttle Model.

This chapter is an overview of implementation of such simulation network constituting the Space Shuttle Model and the Scrub Model. The chapter starts with the current structure of Shuttle Model, then describing the algorithm to develop a RTI compliant Space Shuttle Model using the software adapter provided by the NIST. Implementation procedure is followed then. Finally, the model is tested by the implementation of HLA based simulation network between the Space Shuttle Model and the Scrub Model. The results of the HLA compliant Shuttle Model are analyzed comparing with the results of the HLA non-compliant Shuttle Model, for the same set of run setup parameters.
4.2 Shuttle Model Flow

The figure above is an overview of the hardware flow of a Space Shuttle. Simplifying the structure presented above, the mission flow starts with orbiter processing that occurs in the Orbiter Processing Facility (OPF). The orbiter is prepared for the next mission after detailed inspections, configurations and testing. Then the orbiter is sent to the Vehicle Assembly Building (VAB) where the external tank is attached to it. Finally, the orbiter, as a part of Integrated Space Shuttle Vehicle, goes to the Launch Pad where the preparations for the launch are done. Approximately, 45 percent of the time, the launch is not done. The reasons may be weather delays or technical problems. This part
of the shuttle operational life cycle is the scrub logic. After the shuttle is clean for the launch, the launch is done and it enters the earth orbit to perform its mission. After its mission is executed, the shuttle returns to the earth and after landing, reaches the OPF to follow the cycle again [13].

![Figure 24: Shuttle Process Flow](image)

**4.3 Development Procedure**

The scrub logic from the Figure 24 is to be separated from the Space Shuttle Model. An existing Scrub Model, also developed in arena, has to be used instead of it and the communication between these two arena models is to be done by HLA/RTI using the DMS adapter. This section explains a step by step procedure to achieve the required tasks. The implementation of each step is explained in the following sections.

DMS adapter simplifies the process of developing a HLA compliant simulation model using arena through a set of methods. These methods must be invoked in order to translate the model through different states of the adapter. Arena can invoke the library functions using its visual basic application block.
In the above figure, the Space Shuttle Model is without the scrub logic at its launch pad. Instead, the model communicates with a Scrub Model that has the scrub logic using DMS adapter and HLA. For this, the model must own an adapter, then maintain objects and messages for communication and delete the adapter after execution.

**Step1: Time Management.**

Time management is the logic concerned with the mechanisms for controlling the model in simulation time. If the local simulation time is less than the global simulation time, and if the gap is larger than the simulation step size, the logic advances the model.
with the simulation step size. If the gap is smaller than the step size, then it advances the model by the gap. This logic must be implemented along with the Space Shuttle Model.

**Step2: Initialize the adapter.**

During the pre-initialization state of the model, the model must initialize the adapter i.e., the model is started only after the request for execution is made. The adapter properties are to be set according to the execution conditions. The model creates a federation and waits for other models to join the federation before it starts its run.

**Step 3: Call the adapter for launch approval.**

The model starts in the running mode of the adapter. A shuttle completes its processing through OPF and VBA and reaches the Launch Pad. Here is where a scrub logic is needed and hence a call to the DMS adapter for the approval. The model with the help of Visual Basic Application (VBA) block creates an object with the shuttle and the time information. The object is shared with the Scrub Model and the orbiter in the Space Shuttle Model waits for approval from the Scrub Model. A message is sent to the Space model about the launch approval. Then the orbiter continues with its launch. Other orbiters waiting for the scrub follows the same logic.

**Step 4: Terminate the adapter.**

After the successful execution of the model, it terminates its run. But before terminating it should terminate the adapter.

**4.4 Implementation**

The development procedure described in the previous section is implemented in this section. The section describes the technical details of the model in the steps defined above.
Step 1: Time management:

As shown in Figure, one entity is created at zero time. It invokes the Visual Basic code contained in the VBA block, and delays for an amount of time “a_time” determined from the Visual Basic code. This entity continues this procedure until the simulation is terminated. The pseudo code contained in the VBA block is shown in Figure 27.

```vbnet
If model_simulation_time <= possible_simulation_time Then
    If control_value = 0 Then
        rt = theAdapter.AdvanceSimulation
        control_value = 1
    End If
    If (possible_simulation_time - model_simulation_time) > theAdapter.GetProperty("SimulationStepSize") Then
        s.EntityAttributes[ActiveEntity, temp] = theAdapter.GetProperty("SimulationStepSize")
    End If
    If model_simulation_time = possible_simulation_time Then
        control_value = 0
    End If
Else
    While theAdapter.SimulationAdvanceCompleted <> 1
        DoEvents: DoEvents: DoEvents
    Wend
End If
```

Figure 27: Time Management Pseudo Code

The first “if” condition checks whether the time of the local simulation is behind the current time of the global distributed simulation. If this gap G is larger than the
simulation step size $S$, then it advances the local simulation by $S$. If $G$ is smaller than $S$, then it advances the local simulation by the $G$. In the latter case, the local simulation time becomes equal to the global distributed simulation time. Note that time advancement in the local simulation is performed by specifying “a_time” value and delaying the simulation for “a_time” amount of time. When necessary, the VBA block halts the local simulation until the simulation advance request has been completed. In other words, the local simulation needs to wait physically until all of the other legacy simulations within the same federation catch up to the current time of the global distributed simulation.

**Step 2: Initialize the adapter.**

Arena simulation itself provides a set of visual basic functions that are called at different states of its execution. ModelLogic_RunBegin() is one such method that is called when the model starts its execution. The model in its pre-initializing state initializes the adapter and changes the state to Initialize. Figure 28 presents the pseudo code for the method.
Private Sub ModelLogic_RunBegin()
    Dim rc As Long
    Set m = ThisDocument.Model
    Set s = m.SIMAN
    Set theAdapter = New adapter
    rc = theAdapter.Initialize("init_model.txt")
    Dim rc2 As Long
    rc2 = theAdapter.SetProperty("SimulationStepSize", "72")
    rc = theAdapter.SetProperty("DebugMode", "Enabled")
    rc = theAdapter.SetProperty("SimulationName", "LaunchPad")
    rc = theAdapter.AdvanceSimulation
    If rc < 0 Then
        adapter = False
        m.End
    Else
        adapter = True
    End If
    control_value = 0
    m.QuietMode = True
    MsgBox ("Model 1 has been connected!")
End Sub

Figure 28: Pseudo code to initialize the adapter

The adapter method Initialize() is invoked to start the adapter. Init_model.txt is the input file that contains the properties of the adapter that are to be set. A screen shot of the file is presented in the Figure 29.

<?xml version="1.0" encoding="utf-8"?>
<InitializationFile>
<Properties>
<InitialSimulationTime>0</InitialSimulationTime>
<SimulationStepSize>72</SimulationStepSize>
<SimulationName>LaunchPad</SimulationName>
<DebugMode>Enabled</DebugMode>
</Properties>
</InitializationFile>

Figure 29: Input file data for initializing the adapter

The tags represent the adapter properties and the value between them is the value for that property.
The adapter is in Initialize state until the first call to AdvanceSimulation() method. After the call is made at rc = theAdapter.AdvanceSimulation(), the adapter is at ReadyToRun state and begins its transition to Running state.

**Step 3: Call the adapter for launch approval.**

The request for approval has to be made in the part of the simulation where the previous scrub logic existed. The orbiter arrives at the station “Stacking ready” after OPF and before leaving to the pads. At this point it enters into VBA to wait until the time the Scrub Model delays the orbiter based on its conditions. To process one orbiter at a time, a resource is needed for the orbiter. Hence other orbiters are in the Delay queue until the resource is available. The VBA block calls the method VBA_Block_2_Fire().
Private Sub VBA_Block_2_Fire()
    Set m = ThisDocument.Model
    Set s = m.SIMAN

    Dim shuttleID As Integer
    Dim tempMsg As String
    Dim currTime As Long

    shuttleID = s.EntityAttribute(s.ActiveEntity, s.SymbolNumber("Shuttle Number"))
    currTime = s.RunCurrentTime

    tempMsg = "<Request4Launch>" + 
        "<shuttleID>" + Str(shuttleID) + "
        "<RequestTime>" + Str(currTime) + "
        "</Request4Launch>"

    msgToSend = tempMsg
End Sub

Figure 31: Pseudo Code for constructing a message with shuttle information

In this method, a message is built with the shuttle number and request time. This message is used by the time management block to create an object.

If i > 1 And ReadyToProcess = 1 And RequestSent = 0 Then

    objectID = "91065"
    temp_string = msgToSend

    rc = theAdapter.CreateObject(temp_string, objectID)
    rc = theAdapter.GetObject(objectID, temp_string)
    rc = theAdapter.ReleaseObject(objectID)
    temp_string = "Request4Launch" + "$" + objectID + "$"
    rc = theAdapter.SendMessage("Req", temp_string, "MissionControl")

    s.VariableArrayValue(s.SymbolNumber("RequestSent")) = 1

    msgToSend = ""
End If

Figure 32: Pseudo Code for creating an object

An object is created with an objectID and the message built previously. The object is then released to the simulation network. Then a message is sent to a simulation called as "MissionControl" which is also the Scrub Model with objectID of the object created. This
method updates the message queue of the Scrub Model. The Hold after the VBA block makes sure that the shuttle will wait until the message for launch approval from the Scrub Model is received. This logic is described in the Figure 33.

```vbnet
While theAdapter.AllMessagesReceived <> 1
    msgReceive = theAdapter.GetNextMessage

    If Right(msgReceive, 6) = "</App>" Then
        objID_start = InStr(msgReceive, ";")
        objID_end = InStr(objID_start + 1, msgReceive, ";")
        objectID = Mid(msgReceive, objID_start + 1, objID_end - objID_start - 1)
        rc = theAdapter.SeizeObject(objectID)

        ' need to match
        rc = theAdapter.GetObjectValue(objectID, "/Approval2Launch/shuttleID", shuttleID_str)
        rc = theAdapter.GetObjectValue(objectID, "/Approval2Launch/ApprovalTime", Hours2Launch)
        rc = MsgBox("Approval " + shuttleID_str + ": " + Chr(10) + Chr(13) + "Launch:" + Hours2Launch, vbInformation, "Launch Approved")

        rc = theAdapter.DeleteObject(objectID)
        shuttleID = Val(shuttleID_str)

        ' send signal to module to proceed
        Dim newEntity, symNumber As Long
        newEntity = s.EntityCreate
        symNumber = s.SymbolNumber("Station 165")
        s.EntitySendToStation newEntity, symNumber

    End If
End While
```

Figure 33: Pseudo Code for receiving and decoding a message

The time management block checks for messages at “a_time” delays. If any message is present in the message queue, the message is received and the information for shuttle name and its launch time are encoded. An entity is created to route it to the station to remove the hold on the shuttle as it is ready to launch and the resource is made for other orbiters to enter the VBA block for launch approval. The logic is repeated for each shuttle in the model until the model rests.
Step 4: Terminate the adapter.

Arena calls the Visual Basic function ModelLogic_RunEnd() before the end of each run. This method is used to terminate the adapter.

```vbnet
Private Sub ModelLogic_RunEnd()
    Dim rc As Long
    If adapter = True Then
        rc = theAdapter.Terminate
    End If
End Sub
```

Figure 34: Pseudo Code for terminating the adapter

4.5 Execution Procedure

This section describes the procedure that should be followed to execute the simulation models. RTI and DMS adapter must be installed as per the guidelines, in all the computers. Then implement the following steps,

1. Create a directory `C:\temp\models\` and copy shuttleModel.doe, init_model.txt to it.

2. If executing both the models on a single computer, copy scrubModel.doe and init_model2.txt to `C:\temp\models\`. If on a different computer, create this directory and copy scrubModel.doe and init_model2.txt to it.

3. Run rtiexec.bat located in that computer. If the models are executed on different computers, verify that the value for “multicastDiscoveryEndpoint” in the rtiexec.bat file should match with its value in the RTI.rid file, located in the same computer.

4. Open shuttleModel.doe in Arena.
5. Open scrubModel.doe in Arena. If executing on the same computer, use the “–Multinst” command to open second window of Arena.

6. Run the Arena model - shuttleModel.doe. Wait till a message box saying “Model1 has connected” shows up. Do not click ‘OK’ on that message box.

7. Run the Arena model - scrubModel.doe. Wait till a message box saying “Model2 has connected” shows up. Do not click ‘OK’ on that message box.

8. Click ‘OK’ on both the message boxes.

9. The simulation should now run for one year.

4.6 Execution Details

This section attempts to visualize the execution of the distributed simulation structure following the required procedure. It contains the screenshots of the models in execution mode and the RTI handling its federations.

Before execution of the models an instance of RTIExec is to be initiated to get the HLA.

```plaintext
rtiexec.exe –multicastDiscoveryEndpoint 224.6.15.58:12321
```

![Screenshot of RTIExec](image35.png)

Figure 35: Screenshot of RTIExec
The HLA is ready. The Space Shuttle Model is set to a run setup with 1 replication for an year where the base time units are hours.

![Run Setup](image)

Figure 36: Run setup of the Space Shuttle Model

Both the Space Shuttle Model and the Scrub Model are executed. The Shuttle Model starts its run with initializing the adapter. A federation creation is attempted by both the models. A federation creation success is attained by only one model and it joins the federation. The other model gets a federation creation failure joins the federation created by other simulation. The adapters for the simulations are initialized and the simulation is asked to advance. A message is displayed in a text box that the model is connected to the RTI. Figure 37 depicts the status of the Shuttle Model and HLA.
Figure 37: Federates are Connected to the RTI
Both the federates, MissionControl and LaunchPad are now connected to the RTI. The message box appears on both the simulations and both the simulations are on hold until the messages are responded. When the response from both the messages appear, the adapter transfers to running mode and the model starts its execution.

In the time management block, an entity is created at 0.0 time and it starts checking for messages to be sent or received. This block advances the simulation at intervals. When an orbiter reaches the VBA block in the scrub logic, a message for MissionControl is prepared with the shuttle name and current time. The time management block picks the message up and creates an object with it. A messages is sent to the Scrub Model for the launch approval.

Figure 38: A message is received by the Scrub Model
The message box displayed confirms that the message has been sent to the Scrub Model by the shuttle numbered 2 at 1913\textsuperscript{th} hour. The Scrub Model executes and calculates the delay required and holds the model for that delay. After the delay, the model sends an message to the Space Shuttle Model approving the launch.

Figure 39: A message is received by the Shuttle Model for launch approval

The message box confirms that the shuttle numbered 2 got an approval for launch at 2312\textsuperscript{th} hour which means that the shuttle has been delayed its launch for 399 hours. Once the shuttle is launched the successive shuttles, are handled the same way. The simulation continues its execution and terminates its adapter just before it ends its run. The adapter contains the log of its status throughout the simulation. It contains the
initializing information with the set properties and the simulation run time, simulation steps and federation shutdown.

Figure 40: Federation Shutting down

Figure 40 displays that federates are resigning and the federation execution is shutting down.

4.7 Analysis

The siman report is generated by the Space Shuttle Model after each run. The results after the distributed simulation execution are similar to the results when the shuttle simulation was monolithic. Both the models launched around 7 flights in an year. Only that the scrub logic is replaced with the Scrub Model with HLA communication network.
4.8 Synopsis

In this chapter we discussed the development, implementation and execution of the distributed simulation network with Space Shuttle Model in it through HLA using the DMS adapter as the middleware. This network is a beginning for the development of a huge model, Virtual Test Bed with many existing simulation models replacing similar modules in the Shuttle Model.
CHAPTER 5: CONCLUSIONS AND FUTURE RESEARCH

This chapter provides conclusions of the results obtained from the use of DMS adapter for the Space Shuttle Model to connect to RTI. Further, this chapter discusses the future scope of enhancements to the model.

5.1 Conclusions

The distributed simulation architecture used in this thesis is the High Level Architecture. The research focuses on its goal to use DMS adapter for the legacy simulation model to connect to the RTI in the distributed simulation network environment. It establishes the necessity and working of the DMS adapter with an example in the spaceport operations. The Space Shuttle Model is connected to the RTI with the software adapter as middleware. Understanding the library of functions provided by the adapter, the Shuttle Simulation is connected to the RTI with ease. Since Arena does not support HLA, DMS adapter provides a better option that reduces the necessity to code and making the process less complex. The adapter is simple to expertise and no thorough expertise in HLA tools is required. The use of the adapter is explored with Arena in this thesis, but the same approach can be followed for any COTS simulation languages. Though the simulations used here are developed independently at different times, their purposes are integrated through the RTI for a spaceport operation.

The framework implemented is devoted to a module of spaceport operations. The same framework can be implemented for other modules of the spaceport operations and also be ported to various disciplines like supply chain and distributed manufacturing.
5.2 Contributions

DMS adapter is based on the requirements from the development of Virtual Test Bed (VTB) project, to develop a huge simulation network using many existing simulations totally utilizing the purpose that were developed. This thesis contributes by providing the usage of DMS adapter for the legacy simulations. It also contributes by developing a module for Virtual Test Bed through which, remaining modules can be developed.

5.3 Future Work

Though the DMS adapter reduces the complexity for a COTS simulation language to connect to the RTI, there are still some issues. The interaction requires a lot of steps like developing a common data model, developing interface for the distributed simulation network, adjusting the existing simulation code for it to be a part of the distributed simulation etc.,. Only if the providers of COTS simulation language make their software HLA-Compliant, these issues can be solved. There are some existing HLA languages like SPEEDES, AnyLogic, and MODSIM that are HLA-Compliant. Such languages can be approached for development of new simulations. However, considering the legacy simulations developed in various languages, the adapter provides a minimal possible solution for them to connect to HLA/RTI.

The standard specification of HLA allows several implementation decisions open and to be made by the application developers which enables reusability and integrability of existing codes, but often leaves developers of new simulations without proper guidance. The current usage of the web makes it necessary for the HLA to shift from intranet based networks to internet based networks. Web-HLA is a new form of HLA that
use the emergent standards of distributed computing such as CORBA, COM, JAVA and W3C WOM [7].

The distributed framework developed performs its duties as per required, but it is difficult to monitor the network. Proper visualization tools can be developed to view the systems that are connected to the RTI, exchange of messages and objects, maintain a log of every activity undergone in the network etc.

The DMS adapter implements a time-stepped synchronization approach. Messages are only received when the federate is in the time advancing state. A federate may make the RTI service call: *Enable Asynchronous Delivery* to allow RO messages to be delivered any time *tick* is called, regardless of the state of the federation. DMS adapter does not provide services for *tick*-based asynchronous message delivery. A separate implementation for invoking this RTI service can be developed and used it along with the DMS adapter for achieving a partially asynchronous distributed simulation. Event-based distributed simulation can be approached for a fully asynchronous distributed simulation. This approach maintains a future event list. Every process uses its own local clock. Event-list mechanism causes several events to take place simultaneously. Deadlocks may arise in this approach and hence optimization algorithms are required when building an asynchronous distributed simulation [25].
REFERENCES

1. Andrei Borshchev, Yuri Karpov, Vladimir Kharitonov, "Distributed Simulation of Hybrid Systems with AnyLogic and HLA", St. Petersburg Technical University, Russia


Adapter Reference Guide”


11. Geoffrey C. Fox, Ph. D., Wojtek, Furmanski, Ph. D., Subhash Nair, Hasan T. Ozdemir, Zeynep, Odcikin Ozdemir, Tom A. Pulikal, "WebHLA - An Interactive Multiplayer Environment for High Performance Distributed Modeling and Simulation", Northeast Parallel Architectures Center, Syracuse University


15. Gunther Seliger, Dirk Krutzfeldt, "On the HLA and Internet-based Coupling of Commercia Simulation Tools For Production Networks”, Institute for Machine Tools and Factory Management


25. K. M. Chandy, J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations", University of Texas, Austin


29. Mansooreh Mollaghasemi, Ghaith Rabadi, "Simulation Modeling and Analysis of Space Shuttle Flight Hardware Processing", University of Central Florida

30. Martin ADELANTADO, Muriel BRUNET, "SENSOR MODELING FOR AIRPORT SIMULATION THROUGH THE HIGH LEVEL ARCHITECTURE", French Aeronautics and Space Research Center

31. Nikolai Mansurov, Djenana Campara, "EXTRACTING HIGH-LEVEL ARCHITECTURE FROM EXISTING CODE WITH SUMMARY MODELS"

33. Pitch Technologies, PRTI, “Runtime Infrastructure for HLA 1.3 Simulations”,
   http://www.pitch.se/prti/default.asp

34. Richard M. Fujimoto, "Time Management in the High Level Architecture",
    Georgia Institute of Technology

35. Robert Lutz, "FEDEP V1.4: AN UPDATE TO THE HLA PROCESS MODEL",
    Proceedings of the 1999 Winter Simulation Conference

36. Sean Reilly, Howard Williams, "IDL4HLA: Implementing CORBA IDL Middleware for HLA",
    Anteon Corporation

37. Simon J. E. Taylor, Boon Ping Gan, "HLA-CSPIF PANEL ON COMMERCIAL OFF-THE-SHELF DISTRIBUTED SIMULATION",
    Proceedings of the 2003 Winter Simulation Conference

38. Steffen Straßburger, Thomas Schulze, Ulrich Klein, James O. Henriksen,
    "INTERNET-BASED SIMULATION USING OFF-THE-SHELF SIMULATION TOOLS AND HLA",
    Proceedings of the 1998 Winter Simulation Conference

39. The Distributed Manufacturing Simulation Adapter Reference Guide

40. Thomas Schulze, Steffen Straßburger, Ulrich Klein, "Migration of HLA into Civil Domains: Solutions and Prototypes for Transportation Applications"

41. U.S. Department of Defense, "High-Level Architecture Rules Version 1.3"

42. Wayne J. Davis, Gerald L. Moeller, "THE HIGH LEVEL ARCHITECTURE: IS THERE A BETTER WAY?",
    Proceedings of the 1999 Winter Simulation Conference

44. Xiaojun Shen, Ramsey Hage and Nicolas Georganas, “Agent-aided Collaborative Virtual Environments over HLA/RTT”, University of Ottawa, Canada