TRANSPORTATION SYSTEMS MODELING

USING THE HIGH LEVEL ARCHITECTURE

A Dissertation

by

SHARIF MELOUK

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2003

Major Subject: Industrial Engineering

TRANSPORTATION SYSTEMS MODELING

USING THE HIGH LEVEL ARCHITECTURE

A Dissertation

by

SHARIF MELOUK

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Robert E. Shannon (Chair of Committee)	
Cesar O. Malave (Member)	Don R. Smith (Member)
Robert A. Davis (Member)	Brett A. Peters (Head of Department)

August 2003

Major Subject: Industrial Engineering

ABSTRACT

Transportation Systems Modeling

Using the High Level Architecture. (August 2003)

Sharif Melouk, B.S., Oklahoma State University;

M.B.A., Oklahoma State University

Chair of Advisory Committee: Dr. Robert E. Shannon

This dissertation investigates the High Level Architecture (HLA) as a possible distributed simulation framework for transportation systems. The HLA is an object-oriented approach to distributed simulations developed by the Department of Defense (DoD) to handle the issues of reuse and interoperability of simulations. The research objectives are as follows: (1) determine the feasibility of making existing traffic management simulation environments HLA compliant; (2) evaluate the usability of existing HLA support software in the transportation arena; (3) determine the usability of methods developed by the military to test for HLA compliance on traffic simulation models; and (4) examine the possibility of using the HLA to create Internet-based virtual environments for transportation research. These objectives were achieved in part via the development of a distributed simulation environment using the HLA. Two independent traffic simulation models (federates) comprised the environment (federation). A CORSIM federate models a freeway feeder road with an on-ramp while an Arena federate models a tollbooth exchange.

ACKNOWLEDGMENTS

I would like to express many thanks to my advisor, Dr. Robert E. Shannon, for his support, guidance, and encouragement throughout the duration of this research effort. I appreciate his efforts to make this manuscript more readable and his advice as I prepare to begin my professional career. It is an honor to be the last doctoral student in Dr. Shannon's highly successful career.

I would like to extend my appreciation to Drs. Robert A. Davis, Cesar O. Malave, and Don R. Smith for serving on my committee during this research. I appreciated their input throughout the process especially while writing the dissertation. I am also thankful to Mr. Mark Henry, Mr. Mark Hopcus, and Mr. Dennis Allen for making themselves available to attend to any computer software and/or hardware issues during the testing phase of this research.

A special thanks goes to Ashwin Kekre, Hardik Parekh, and Dilip Musani. They were instrumental in the development of the simulation environment, especially with respect to interface development. I greatly appreciate their efforts and ideas during this critical portion of the research effort. Last, but not least, I would like to thank my family and all my friends at Texas A&M University for their support.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

This dissertation investigates the High Level Architecture (HLA) as a possible distributed simulation framework for transportation systems. Section 1.1 gives the background and research motivation. Section 1.2 describes the research specifications in terms of the purpose, objectives, and goal. Section 1.3 discusses the method of approach for achieving each of the objectives. Section 1.4 presents the organization of the dissertation.

1.1 Background and Research Motivation

In traffic engineering, the concept of traffic control is giving way to the broader philosophy of Transportation Systems Management (TSM) whose purpose is to optimize the utilization of transportation resources and improve the movement of people and goods without impairing the community. The economic importance of traffic management grows each day. Well designed and well managed highway systems reduce the cost of transporting goods, cut energy consumption, and save countless hours of driving time. The state-of-the-art in transportation engineering has advanced dramatically over the past decade with the emergence and application of new, more flexible traffic control devices, software systems, computer hardware, communications

---

This dissertation follows the style and format of *Transportation Science*.

and surveillance technologies, and analysis methods. However, only recently has the state-of-the-art been able to tap the full potential of these advances to effectively address and resolve transportation issues within existing transportation infrastructures.

One of the most important analytical tools of traffic engineering and transportation systems management is computer simulation. When a traffic system is simulated, it is possible to predict the effect of traffic control and traffic systems management strategies on the system's operational performance. Typical measures of effectiveness are average vehicle speed, vehicles stops, delays, vehicle-hours of travel, fuel consumption, and pollutant emissions.

TSM simulation models are designed to represent traffic in a particular physical environment (streets and freeways) and at a specific level of simulation detail (microscopic or macroscopic). Traffic can be viewed as a complex system, and developing macro models is one of the primary approaches to modeling complex systems. Macro models follow a top-down approach, focusing on the observable behavior of a system in terms of aggregate, abstract parameters and their probability distributions. In the case of traffic, macro models are usually derived from fluid dynamics and involve aggregate parameters such as traffic volume and average speed on arteries in a traffic network. Simulation-based macro models have the advantage that run-time can be relatively short, as the computation is based on aggregate, abstract parameters. Macroscopic models are helpful when a coarse prediction of conditions is

sufficient. However, most aspects of complex systems are highly nonlinear. Such systems are often extremely sensitive to initial conditions, and even very small perturbations in initial conditions can have a very large impact on the global system behavior. In the process of aggregating and abstracting information, macro models lose their sensitivity and capture the behavior of traffic only under idealistic conditions. An alternative approach that can potentially produce better results is micro modeling. This is a bottom-up approach, where a complex system is viewed as a large set of small, interacting components. The main focus is on identifying the components of the system, discovering their local behaviors, and the interactions among them. The global system behavior emerges from the local behaviors of the individual components and their interactions. Developing microscopic simulation models represent movements of individual vehicles including the influences of driver behavior.

Traditionally, traffic simulation models have been developed as monolithic, stand-alone systems that are well suited for the purpose for which they were designed. Microscopic traffic simulation models are typically characterized by a high degree of detail. These monolithic models contain many static components of the infrastructure (e.g. streets, intersections, traffic lights etc.), decision rules, and dynamic entities like vehicles, pedestrians and bicycles. The number of components and the number of relations between components lead to large-scale models. Often, microscopic traffic simulation models become too large, unwieldy, and difficult to maintain and adapt. These circumstances lead to:

- Long run times (hours) for simulation runs
- Long times for developing and testing of such monolithic models
- Excessive human effort to maintain models and adapt models for other purposes
- Low flexibility and reusability

One solution is the employment of more powerful hardware. Another is breaking up the model into a distributed set of sub-models and using distributed computing. Different approaches exist for how a monolithic simulation task can be divided into sub-models. The two main approaches are:

(A) Every component describes a different functional subsystem. A traffic simulation could consist of separate models for vehicle traffic, traffic light control, and pedestrian traffic.
(B) Every component describes a complex independent model for a small geographic region.

By far the most promising approach to distributed simulations is the High Level Architecture (HLA) being developed by the Department of Defense under the Defense Modeling and Simulation Office (DMSO). The High Level Architecture (HLA) is a major ongoing effort of the Department of Defense (DOD) to define a standard specification of a common technical architecture for use across all classes of DOD simulations. Begun in 1995, the goal is reusability and interoperability. By this they mean the ability to take stand alone, monolithic simulations developed for a particular purpose and put them together in various combinations (federations) for the study of new, more complex problems. Some of the DOD motivation was for large war games and training simulations. Although developed for use in the military arena, the HLA

appears to have great potential for use in civilian applications and, in particular, traffic management problems.

The HLA does not prescribe a specific implementation, nor does it mandate use of any particular hardware, software, or programming language. It is a form of distributed computing where the different simulations, databases, and human decision makers (also called viewers) can be on different machines and in different geographical locations. The importance of distributed simulations has increased. Because of this, there have been several efforts in the commercial sector to enable distributed computing. Two of the most viable recent efforts are the Common Object Request Broker (CORBA) by the Object Management Group, and Remote Method Invocation (RMI) from Sunsoft's Java Development kit (Buss and Jackson, 1998). Each of these architectures for distributed computing offers much to the problem of distributed simulation. However, the HLA was chosen for the following reasons: (a) Both CORBA and the HLA are concerned with legacy applications (federates), possibly in different languages, i.e. they are language neutral while RMI federates must be written in JAVA. (b) CORBA and RMI are oriented toward general applications whereas the HLA is specifically targeted at distributed simulations. Thus, the HLA provides more powerful support for simulation issues such as time management. (c) The HLA allows transfer of object ownership between federates whereas CORBA and RMI do not. For situations involving legacy simulation models written in different languages and running on different hardware, the

HLA provides more than the other two, in large part due to its simulation related services.

Although it is just now being explored, the HLA seems to be suited for civilian applications. The coupling of geographically, organizationally or otherwise distributed systems, simulations, viewers, information systems etc. is a feature especially interesting for traffic management applications.

Much like any computer programmer, each simulation modeler has their own modeling style, which leads to unique modeling approaches even when the same type of environment, such as transportation, is being modeled. This is a significant problem when it would be beneficial for two or more of these models to communicate. For example, modelers can have different interpretations of object classes and their corresponding attributes and interactions. One modeler may refer to all vehicles as "cars" while another modeler may classify vehicles as "cars", "trucks", and "buses". These differences in naming convention can also exist with the attributes and interactions. These elementary types of inconsistencies make it difficult to determine whether certain simulations are viable candidates for the same distributed environment. This problem would be eliminated if each simulation were HLA-compliant.

The HLA standard not only defines the types of services it provides and how it will provide them, but it also calls for standardization in terms of documentation. This

documentation completely defines each simulation (federate) in terms of its object classes, attributes, interactions, parameters, etc. Everything is explicitly documented, including naming convention, so that federation developers can determine if a particular federate's offerings are suited for their federation execution. Via this documentation, federation developers are able to look at the basic structure of foreign federates, but they are shielded from viewing potential proprietary information, such as the simulation source code itself. The ease and secure nature in which federates can be reviewed leads to increased collaboration among simulation creators, which may lead to distributed transportation environments being developed that otherwise would not have. A set of HLA-compliant simulations essentially creates a "plug and play" situation. Once viable candidates for a distributed simulation execution are identified, the participating federates are linked and begin to interact with little modification. This allows for federates to be swapped with one another to create alternate federations and contributes to the analysis of several different scenarios of a transportation system.

1.2 Specifications of the Research

This section describes the purpose, objectives, and goals of the research.

1.2.1 Purpose

The purpose of this research is to determine the viability of the High Level Architecture as a framework for transportation system models.

1.2.2 Research Objectives

The research objectives are as follows: (1) determine the feasibility of making existing traffic management simulation environments HLA compliant; (2) evaluate the usability of existing HLA support software in the transportation arena; (3) determine the usability of methods developed by the military to test for HLA compliance on traffic simulation models; and (4) examine the possibility of using the HLA to create Internet-based virtual environments for transportation research.

1.3 Method of Approach

The HLA concept and existing tools and method used for implementation were studied with respect to the development of a distributed transportation environment. The initial step in the development was to identify and evaluate the most widely used transportation simulation software packages for their ability to have the concept of the High Level Architecture implemented upon them. Two transportation simulations were created using CORSIM version 3.2 and Arena 4.0, respectfully. These software packages were chosen for their widespread use throughout the simulation community, and their ability to communicate with external software via an interface structure already present within the software. These simulations comprise the transportation simulation environment used to explore and demonstrate the HLA. The CORSIM model simulates a freeway feeder road, and the Arena model simulates a tollbooth exchange. Vehicles created in the CORSIM model are transferred to the Arena model using the HLA services.

Each simulation model had an interface developed for it so the models could communicate with one another. Each interface is essentially a bridge that allows each model to exchange information with the Run-Time Infrastructure (RTI), the "operating system" of the HLA. The RTI acts as a vehicle for information exchange among the individual simulation models comprising the entire simulation environment. CORSIM's Run-Time Extension (RTE) feature was used to develop the CORSIM model's interface. Arena's external subroutine capabilities were used to develop that model's interface.

Existing tools and methods were employed throughout the development of this traffic simulation environment. These utilities, including an HLA compliance testing procedure, are evaluated with respect to their usefulness in transportation simulations.

1.4 Organization

This dissertation is organized into six chapters. Chapter II presents the literature surveyed in the areas of distributed traffic simulation, web-based simulation, and the High Level Architecture. Chapter III presents background information on the High Level Architecture. Chapter IV details the two simulation models comprising the traffic simulation environment. The procedure to implement the HLA functionality is described for each model. Chapter V includes an evaluation of existing tools, methods, and procedures used for the HLA implementation. Additionally, there is a discussion of the HLA with respect to web-based simulation. Chapter VI is a final discussion and gives future research directions.

CHAPTER II

LITERATURE REVIEW

2.1 Introduction

HLA is a very young science. Therefore, the amount of research done in this area is not extensive, especially in relation to transportation systems modeling. However, some researchers have contributed to the growing areas of distributed traffic simulation and web-based simulation. This chapter surveys the relevant contributions on these topics as well as studies on the HLA. The following three sections highlight some of the important results published in these areas.

2.2 Distributed Traffic Simulation

Klein, Schulze, and Straβburger (1998) investigated traffic simulation based on the HLA. Their work included enhancing classic simulation and animation tools for HLA compatibility. It was also shown that legacy simulation models, independently extended for HLA compatibility by different organizations, could exist in a distributed simulation environment. Straβburger (1999) addressed the application of HLA-based coupling of simulation tools. The concepts for the extension of simulation tools with HLA capabilities and their realization are discussed. Schulze, Straβburger, and Klein (1999) discussed the problems of a simulation interoperability standard with civil applications. Solutions to these problems, especially in the area of transportation, were suggested using the HLA. Ozaki, Furuichi, Nishi, and Kuroda (2000) developed a modular,

flexible, and scalable micro-model car traffic simulation system. The HLA was applied to every system module as a standard interface between each module for evaluating and validating a variety of micro-model simulation schemes. Klein (2000) presented a new information technology approach for simulation-based systems capable of serving multiple purposes through the composition of components. Based on the HLA, a prototype dispatching system for public transportation demonstrates the concept.

2.3 Web-based Simulation

The area of web-based simulation is becoming an area of great research interest. Page (1998) explored the relationship between HLA and web-based simulation. The paper discusses whether HLA could serve as an interoperability technology for the commercial and academic sectors in the age of web-based simulations. Potential barriers of the transfer of this DoD technology are illustrated, and mechanisms through which these barriers may be overcome are suggested. Straβburger, Schulze, Klein, and Henriksen (1998) presented approaches by which HLA can be used to interconnect distributed model components which are developed using commercially available, off-the-shelf simulation software. The requirements imposed on the simulation software by HLA are discussed. Page and Opper (2000) described the evolution of web-based simulation and derive a collection of modeling principles that characterize the future of web-based simulation. These principles are examined in terms of their implications for next-generation computer generated forces systems, a term used to describe simulations that provide representations of military forces. Miller et. al (2001) reviewed the

development of research in the area of web-based simulation. Potential opportunities and areas of focus are identified in both academic and industrial arenas. Kuljis and Paul (2000) discussed potential environments and languages for web-based simulation. They concluded that web-based environments would continue to be isolated efforts as simulation developers continue to resist newer simulation advancements.

2.4 High Level Architecture

There are several topics related to the HLA that are fertile research areas. Some of the early works bearing relation to this research effort are presented below.

Pratt and Dugone (1999) presented a step-by-step guide for making legacy distributed interactive simulation (DIS) HLA compliant using the HLA Gateway. The guide is geared for developers with limited HLA backgrounds and funding to make their DIS-based simulations HLA compliant, when HLA compliance is needed fast, and when existing DIS capabilities for the simulation are to be maintained.

Buss and Jackson (1998) compared three architectures for supporting distributed computing, HLA, Common Object Request Broker (CORBA), and Remote Method Invocation (RMI). It was concluded that for situations involving legacy simulation models written in different languages, HLA provides more than the other two due to its orientation toward simulation and its simulation-related services.

Horst and Woldt (2000) discussed using tools developed by the Defense Modeling and Simulation Office (DMSO) to facilitate the HLA compliance testing process. Many of these tools allow for federate developers to assess their simulations before submitting them for formal HLA certification. Horst et al. (2000) offered improvements to the HLA federate compliance testing process. Improvements are suggested for the runtime component of the compliance testing process by eliminating the need for a predetermined test sequence. Other upgrades include simplifying the compliance testing process for the Certification Agent by implementing a single federate compliance testing tool, and retooling the compliance testing website to handle the new testing process.

Paterson et al. (2000) describe how gateways and other middleware can aid in the implementation of the HLA on individual simulations. It was found that the middleware software can house several of the RTI services, such as data distribution management and time management, in such a way that the HLA services are hidden. It is also noted that the use of middleware could alleviate some of the cost required to make a simulation HLA compliant. McLean and Fujimoto (2001) discuss the usefulness of the federated-simulations development kit (FDK) in developing customized RTIs for use in a distributed simulation environment. The FDK consists of modules so that RTI developers can select only the modules that are most important for their RTI implementation. Several instances in which the FDK software is used are described. Sauerborn et al. (2000) recommend changes to the ownership management services specification of the RTI. The changes include: allowing attribute transfer to be directed

to a specific federate and providing a federate with the ability to inform the RTI that it is unwilling or unable to accept a particular attribute instance. This paper represents a beginning work in the research and upgrade of the HLA specification, a fertile area for continued research.

Bachinsky et al. (2000) developed a federation engineering approach to distributed simulation exercises. The approach includes system administration, network configuration, and systems engineering. The approach serves as a vehicle to ensure the success of the distributed simulation exercise. Rouget and Henry (2000) studied developing tools to examine any HLA compliant object model and automatically generate HLA federate source code. This source code separates the interface between the RTI implementation and the federate developers application. This reduces the need for developers to have an in-depth understanding of the RTI API, and allows developers to concentrate on their particular simulation application. Pace (2001) discusses the simulation conceptual model's role in determining compatibility of candidate simulations for a HLA federation. Key conceptual model attributes are presented, and suggestions are given to guide the validation of proposed distributed simulation environments. Federate compatibility issues and federation options when federates are incompatible are identified. It is contended that federation compatibility is a key element of federation validation and has a role in effective interoperability. Stytz and Banks (2001) recommend improvements to the design and documentation of distributed simulation environments using the unified modeling language (UML). Aspects of UML

that can be used to complement the federation development process are discussed. Their recommendation is that the majority of the federation and federate design and documentation process be done using the tools provided by UML.

As can been seen, there are numerous areas in which the HLA can be explored and applied. The next chapter introduces the HLA to familiarize the reader with it.

CHAPTER III

THE HIGH LEVEL ARCHITECTURE

3.1 Introduction

The High Level Architecture (HLA) is an object-oriented approach to distributed simulations developed by the Department of Defense (DoD) under the Defense Modeling and Simulation Office (DMSO). More specifically, the HLA is intended to address the issues of reuse and interoperability of simulations. Reusability means that component simulation models can be reused in different simulation scenarios and applications. Interoperability means that the reusable component simulations can be combined with other components without the need for re-coding. Although the HLA was developed for use in military applications, it appears to have great potential for civilian applications including transportation systems modeling.

The HLA is an emerging technology for linking simulations of various types at multiple locations to create a realistic, complex, "virtual world" for the simulation of highly interactive activities. This technology brings together systems built for separate purposes, technologies from different eras, products from various vendors, and diverse hardware at different locations and permits them to interoperate together in a synthetic environment. The HLA establishes a high-level common ground that facilitates the interoperability of several simulation models. It essentially manages how data is distributed among the participating simulations in the distributed environment. Unlike

other previous network architectures, the HLA directs data to simulations on a need-to-know basis only. This allows for reduced data traffic, and therefore, a more efficient distributed simulation environment.

The next section will discuss the components defining the HLA. This is followed with sections detailing the RTI and the process of creating and running a federation exercise.

3.2 Components of the HLA

The HLA is defined by three components: Federation Rules, the HLA Interface Specification, and the Object Model Template. Table 3.1 below provides an overview of these components (DMSO, 1999).

Table 3.1
HLA Components

Federation Rules	• Ensure proper interaction of simulations in a federation • Describe the simulation and federate responsibilities
Interface Specification	• Defines Run-Time Infrastructure services • Identifies "callback" functions each federate must provide
Object Model Template	• Provides a common method for recording information • Established the format of key models: federation object model, simulation object model, and management object model

The Run-Time Infrastructure (RTI) software is the backbone of any distributed simulation system developed within the HLA framework. It is essentially an "operating system" that provides services to the federates (simulations) participating in the distributed environment.

3.2.1 Federation Rules

The federation rules consist of ten rules that describe the conduct of federates and their interaction with the RTI (DMSO, 2000). Five rules relate to the federation and the other five to the federate.

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 RTI.
3. During a federation execution, all exchange of FOM data among federates shall occur via the RTI.
4. During a federation execution, federates shall interact with the RTI in accordance with the HLA interface specification.
5. During a federation execution, an attribute of an instance of an object shall be owned only one federate at any given time.

Federate Rules

6. Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA OMT.

7. Federates shall be able to update and/or reflect any attributes of objects in the SOM and send and/or receive SOM object interactions externally, as specified in their SOM.
8. Federates shall be able to transfer and/or accept ownership of an attribute dynamically during a federation execution, as specified in their SOM.
9. Federates shall be able to vary the conditions under which they provide updates of attributes of objects, as specified in their SOM.
10. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

Federate developers must adhere to these ten rules in order for their federate to be HLA compliant.

3.2.2 HLA Interface Specification

The interface specification mandates how the federates will interact with the federation and each other. The specification is comprised of six management areas: federation management, declaration management, object management, ownership management, time management, and data distribution management.

Table 3.2 below summarizes these management areas (DMSO, 1999).

Table 3.2

Management Areas of the Interface Specification

Management Area	Responsibilities	Example(s)
Federation Management	Control an exercise	• Creating a federation • Joining federates to a federation
Declaration Management	Define data publication and subscription	• Publication of attributes • Subscription of attributes
Object Management	Exchange object and interaction data	• Register objects • Update attributes
Ownership Management	Transfer attribute ownership	• Divesture of an object's attribute(s) • Acquisition of an object's attribute(s)
Time Management	Control message ordering	• Request current time • Advance time of the simulation
Data Distribution Management	Efficiently route data between producers and consumers	• Create a region • Modify a region

Federation Management

This management area contains several services that any HLA compliant federate must be capable of performing. The mandatory services are the following: creating a federation, joining a federation, resigning from a federation, and destroying a federation. All federates attempt to create a federation upon connection to the RTI process. However, only the first federate to connect to the RTI process will be successful in doing

this, while exceptions will be raised with the other federates. The other federate(s) will then perform a join federation execution.

Declaration Management

This management area describes the publication and subscription functions of the HLA. Typically, federates produce object instances or interactions. Each federate must declare exactly what it is able to publish, or generate. Similarly, federates usually consume object instances or interactions. Each federate must declare exactly the instances and interactions in which it wishes to subscribe, or recognize. The RTI keeps tracks of each federate's interests to achieve the goal of efficient communication among the federates.

Object Management

This management area includes registering, discovering, and deleting object instances, and updating and reflecting object attributes. Federates introduce object instances to the federation via a registration process. However, attribute values for the instance are not provided until a separate, second step is performed. This is done via update and reflection methods.

Ownership Management

This management area describes how federates update and delete object instances. Object instances may be wholly owned by one federate that would be solely responsible for updating the attributes associated with the object or for deleting it. Multiple

federates may share responsibility of updating an object's attributes. In this case, each of the participating federates has the responsibility of updating a mutually exclusive set of attributes. Additionally, each participating federate has the right to delete an object instance. Federates may also exchange attribute ownership by attempting to push ownership on another federate or by attempting to acquire ownership of an attribute from another federate.

Time Management

This management area presents the methods the RTI uses to manage time advances during a federation execution. This is an optional service but would mandatory in federations where time synchronization is key. In this research effort, time management is ignored, as it is not necessary to implement in order to achieve HLA compliance.

Data Distribution Management

This management area describes how data can be efficiently routed during a federation execution. Routing spaces are used to further isolate publication and subscription interests. This aids the RTI being used as a switching device to transfer data among the federates. Once again, this is an optional service and was not implemented.

3.2.3 Object Model Template (OMT)

The OMT provides a common method for recording information and establishes the format of key models. These models are the Federation Object Model (FOM),

Simulation Object Model (SOM), and Management Object Model (MOM). The key features of the three models are listed below (DMSO, 1999).

Federation Object Model

- One per federation
- Introduces all shared information (e.g., objects, interactions)
- Contemplates interfederate issues (e.g., data encoding schemes)

Simulation Object Model

- One per federate
- Describes salient characteristics of a federate
- Presents objects and interactions which can be used externally
- Focuses on the federate's internal operation

Management Object Model

- Universal definition
- Identifies objects and interactions used to manage a federation

The Object Model Development Tool (OMDT) can be used to construct these object models. For this particular application, the FOM for the federation and the SOM for each federate are identical since both federates in the distributed environment have the same object instances and interactions. The OMDT also has a utility to automatically

create the Federation Execution Data (FED) file. The FED file contains initialization information required by the RTI relating to the FOM.

3.3 Run-Time Infrastructure (RTI)

Again, the RTI is the backbone of the HLA. It is software that serves as the "operating system" for the federation by providing essential services for proper interaction. The RTI has three components: the RTI Executive process (RtiExec), the Federation Executive process (FedExec), and the libRTI library.

The RtiExec is a globally known process that manages the creation and destruction of FedExecs. Each federate within a particular federation communicates with the RtiExec to initialize RTI components. The RtiExec also assures that the FedExec has a unique name. This is important because more than one federation (hence, multiple FedExecs) can use the same RTI at the same time. The FedExec manages the federation in that it allows federates to join and resign from a federation. It also oversees data exchange between participating federates. The libRTI is a C++ library that provides the services described in the HLA Interface Specification. These services were discussed in Section 3.2.2.

3.4 Federation Execution

This section describes the step-by-step process of executing a federation exercise.

Step 1

Start the RTI from any terminal with a known IP address so that external federates can locate it and use the services provided by the RTI. This action starts the RtiExec process.

Step 2

A participating federate then begins simulating. This federate will register with the RtiExec and create a federation. This action starts the FedExec process and reserves a federation name with the RtiExec. The FedExec also registers a communication address with the RtiExec.

Step 3

The remaining participating federates can now join the newly created federation. Each federate obtains the FedExec address from the RtiExec and invokes a join federation execution.

Step 4

As each federate finishes its activity within the federation execution, each federate must resign from the federation. When the federates are done executing, a destroy execution is invoked to eliminate the federation.

CHAPTER IV

FEDERATION DEVELOPMENT AND EXECUTION

4.1 Introduction

This chapter details the development of a distributed simulation environment involving two separate traffic simulation models (federates). More specifically, this chapter discusses the accomplishment of the first objective of this research effort: (1) determine the feasibility of making existing traffic management simulation environments HLA compliant. The test environment consisted of federates developed using CORSIM and Arena, respectively. CORSIM, developed by the Federal Highway Administration, is a microscopic simulation software that models surface streets and freeways along with a wide array of traffic control devices. CORSIM is an integration of two microsimulation models, NETSIM and FRESIM, respectively. NETSIM is used to model traffic on urban streets while FRESIM is used to model traffic on freeways. Arena is widely known, commercially available, general simulation software capable of modeling many different types of environments, including a traffic simulation environment.

The CORSIM federate models a freeway feeder road with an on-ramp leading to a freeway tollbooth exchange. The second federate is the tollbooth exchange and is modeled using Arena. The network has no signal control. The vehicles are the only objects defined in the simulation with vehicle attributes of vehicle identification number, status, and exit time. The development of this test environment includes: developing a

separate interface for each federate so each federate can communicate with the RTI and each other, and creating the appropriate support files so as to comply with the HLA standard.

4.2 CORSIM

CORSIM is a continuous traffic simulation software in that it uses a continuously advancing fixed time-step to describe traffic operations. Each vehicle is treated as a distinct entity in the simulation and its state is updated every time-step. A time-step is one second. The state of the vehicle is defined by its position relative to other vehicles and the time-step. A simple network in CORSIM consists of entry nodes, exit nodes, and internal nodes connected by unidirectional links. The links typically represent urban streets or freeway sections, whereas nodes represent urban intersections or points of traffic convergence/divergence.

The input file to CORSIM is a fixed format file that contains information about the network. It specifies both spatial and time-varying characteristics of the network. The input stream consists of a sequence of "block" data records. A block outlines the conditions during one time period and up to nineteen such blocks may be specified. Each block is divided into sections that are further divided into record types. Record types contain specific data items and are divided into 80 columns each. Every column holds pre-determined information about a part of the network or its characteristics. CORSIM has a detailed procedure to test the validity of the input stream.

CORSIM is available in a Microsoft Windows® environment called the Traffic Software Integrated System (TSIS). TSIS provides a user interface to CORSIM and permits it to interact with other software programs. TSIS's CORSIM environment is composed of four major components:

- TSIS.exe – the executable Microsoft Windows® program
- TSISINTF.dll – a dynamic link library that contains shared memory, communication interface and the TSIS Application Programming Interface (API)
- CORSIM.dll – a dynamic link library that is built from configuration-controlled source code
- Run-Time Extension (RTE) – an optional dynamic link library that can be used to interact with CORSIM at runtime

CORSIM and TSIS exchange data by using a shared memory area in TSISINTF.dll. The TSIS environment also controls any external libraries that interface with CORSIM.

CORSIM, developed by the Federal Highway Administration, is used to represent traffic flow along surface streets and freeway systems. CORSIM is an integration of two microscopic simulation components:

- NETSIM – used to model traffic on surface streets
- FRESIM – used to model traffic on freeways

The CORSIM Data Dictionary lists pre-defined variables within the NETSIM and FRESIM frameworks, respectively, that can be used to access information from the CORSIM database. The RTE can import these variables in order to extract information from the simulation. This information can be gathered and used as data for a statistical analysis, or it can be passed on to external programs as necessary input for proper execution. For example, the ENTIME(X) variable retrieves the time in tenths-of-a-second that vehicle X entered the network.

TSIS's CORSIM has three interface functions. These are listed below:

- INIT – an interface function called by TSIS at the start of the simulation
- JMAIN – an interface function called by TSIS once every simulation time-step
- JEXIT – an interface function called by TSIS at the end of the simulation run

The RTE can be configured to include corresponding routines that are called by these functions. These three functions can thus act as entry points for that particular run-time extension.

4.2.1 CORSIM Federate

In order to test ease of compliance of CORSIM simulations to the HLA standards, a small feeder road simulation model was created. The model simulates a freeway feeder road with an on-ramp. The network has no signal control.

Figure 4.1 below depicts this simulation model.

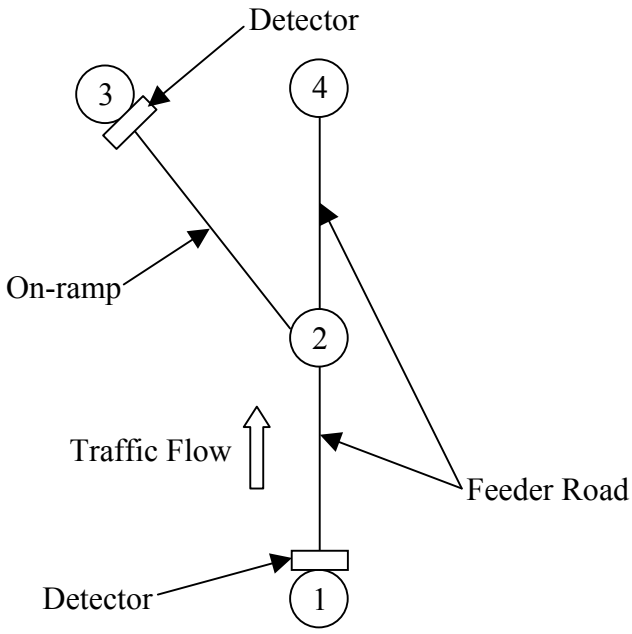


Figure 4.1. CORSIM Federate (Feeder Road with on-ramp)

Vehicles enter the network at node 1. All traffic is one-way; say traveling from "south to north". At node 2, vehicles can turn toward node 3 (the on-ramp) or continue toward node 4 (continuation of the feeder road). It was arbitrarily set that 50% of the vehicles take the on-ramp while the remaining vehicles stay on the feeder road. Vehicles that reach node 4 are disposed while vehicles at node 3 (the end of the feeder road) are "transferred" to the tollbooth exchange model (the Arena federate). Detectors reside at nodes 1 and 3, respectively, and serve as triggering mechanisms during simulation execution. The only objects defined in the simulation are vehicles with attributes of

vehicle identification number, status, and exit time. The change in the value of the status attribute of a particular object (vehicle) is used as an indication of the vehicle's exit from the feeder road. The interoperation between these two simulations can be achieved using the RTI in accordance with the HLA standards.

4.2.2 CORSIM Interface Environment

CORSIM is interfaced with the RTI, and hence the HLA, via the RTE feature existing in CORSIM. Three functions are declared in CORSIM that act as entry points to the RTE. These functions are called upinit, upcntrl, and upexit, respectively. These functions are called by the INIT, JMAIN, and JEXIT subroutines in CORSIM. These subroutines are found in KSC.FOR, a file in CORSIM that provides interface capability for CORSIM. The INIT subroutine calls the upinit C function, the JMAIN subroutine calls the upcntrl C function, and the JEXIT subroutine calls the upexit C function.

A project workspace file, named interfac.dsw, was created to house the source and header files responsible for interfacing with the RTI.

These files are listed in Table 4.1 below.

Table 4.1

Source and Header Files in the CORSIM Interface Environment

Source Files	**Header Files**
binarySequence.cpp	binarySequence.h
corsim_interface_RTI.cpp	detector.h
detector.cpp	FederateAmbassador.h
FederateAmbassador.cpp	Global_Variables.h
integer.cpp	integer.h
lane.cpp	lane.h
link.cpp	link.h
network.cpp	netsim.h
node.cpp	network.h
signalState.cpp	node.h
traf_RTI.cpp	traf_RTI.h
upcntrl_HLA.cpp	upcntrl.h

All the files listed above already exist within CORSIM with the exception of 3 source files and 3 header files. Corsim_interface_RTI.cpp, FederateAmbassador.cpp, traf_RTI.cpp, FederateAmbassador.h, Global_Variables.h, and traf_RTI.h are the added files. Simply stated, CORSIM interfacing with the HLA is the interaction of the files listed in Table 4.1. The following section details an execution of the CORSIM federate.

4.2.3 CORSIM Interface with the HLA

This section describes the interaction between the files listed in Table 4.1. Essentially, a single entity (vehicle) is followed throughout the simulation.

Primary services required for the CORSIM simulation to create and/or join the federation are called from the upinit function. This function also calls the services necessary to get handles to the object attributes and those needed for publish and subscribe mechanisms. All these services are performed when the CORSIM model is first executed and the entry of the first vehicle is detected by the detector placed at node 1 (see Figure 4.1). Every vehicle passing the detector triggers it. The upinit function resides in upcntrl_HLA.cpp (see Appendix 1) and immediately calls another function, InitialFunction(), which is in corsim_interface_RTI.cpp (see Appendix 2). The upinit function code is shown below:

```
void UPINIT( char *fname )
{
        //initialization routine
        //called once at the beginning of simulation
        int i;
        int j;


        char outbuffer[132];
        InitialFunction();

        sprintf(outbuffer,"init done \n");
//      CWRITE(outbuffer,132);

        endOfInit=0;
        prevInit=0;

        pNetwork=new CNetwork();

        //fname must be null terminated
        for (i=2;i<512;i++)
        {
                if (((fname[i-2]=='T')||(fname[i-2]=='t'))&&
                        ((fname[i-1]=='R')||(fname[i-1]=='r'))&&
```

```
                ((fname[i]=='F')||(fname[i]=='f')))
        {
                j=i+1;
        }
    }

    for (i=j;i<512;i++)
    {
        fname[i]='\0';
    }

    pNetwork->m_TrafInputFile=CString(fname);

    //read the traf file
    pNetwork->ReadTrafFile();
    return;
}
```

Within InitialFunction(), calls are made to three other functions in traf_RTI.cpp (See Appendix 3), namely JoinFederation(), GetAttributeHandles(), and PublishSubscribeAttributes(). The InitialFunction() function code is shown below:

```
void InitialFunction()
{
    JoinFederation();
    GetAttributeHandles();
    PublishSubscribeAttributes();
}
```

The JoinFederation() function serves to create a federation called “Feeder Road” or allows the simulation (or federate) to join an existing federation named “Feeder Road”. In the case of the first vehicle entering the network, the function will serve to create the federation. All subsequent federates will join the existing federation. The GetAttributeHandles() function gives the handles for the vehicle class (Vehicle) and the attributes (ID, Status, and CorsimExitTime) for this class. The

PublishSubscribeAttributes() function enables the federate to reveal and gather information regarding each vehicle in terms of its attributes (vehicle identification number, status, and exit time).

For every time step in which the first detector is triggered by a vehicle entering the network, an object of the class 'vehicle' is instantiated in the federation by calling the RTI service responsible for creating an object. Similarly, the departure of a vehicle from this federate is detected using another detector at the end of the on-ramp (the detector at node 3). In the event of a vehicle exiting the system, the value of the status attribute is changed using the RTI service for updating attributes. All these tasks are included in the upcntrl function, in upcntrl_HLA.cpp, so as to obtain an update of the network every second. This function calls another function, CentralFunction(), which is in corsim_interface_RTI.cpp. The upcntrl function code is shown below:

```
void UPCNTRL()
{
        int time;
        BOOL init;
        POSITION pos, detpos;
        CLink* pLink;
        CDetector* pDetector;
        static vector<int>count;

        init=yinit;

        //the algorithm that controlls the signal states at the
        //intersections assumes time is always increasing, but
        //the CORSIM clock starts over after initialization
        //so the time at which initialization is over must
        //be recorded
```

```
	if ((!init)&&(prevInit))
	{
		//end of initialization
		endOfInit=prevTime+1;
	}

	//adjust the time by adding the end of initialization
	time=sclock+endOfInit;

	//get signal state for the node under corsim control
	//pNetwork->UpdateNodeSignalStates();

	//process any detector information

	pos=pNetwork->m_LinkList.GetHeadPosition();
	while (pos!=NULL)
	{
		pLink=pNetwork->m_LinkList.GetNext(pos);


		if(pLink->m_upnode->m_id==1 && pLink->m_dnnode->m_id==2)
		{
			pLink->ProcessDetectors();
			detpos = pLink->m_listOfDetectors.GetHeadPosition();
			pDetector = pLink->m_listOfDetectors.GetNext(detpos);
			count.push_back(pDetector->m_count);
			if(count[count.size()-1]-count[count.size()-2] && time>300) {
/*
				char outbuffer[132];
				char ch[20];
				_itoa (time-300,ch,10);
				sprintf(outbuffer,ch);
				CWRITE(outbuffer,132);

*/
				CentralFunction();
			}
		}

	}
	//record whether the simulation has reached equilibrium
	//or not, so the time at which initialization can be
	//recorded
	prevInit=init;
```

```
        prevTime=time;
}
```

Within CentralFunction(), calls are made to two other functions in traf_RTI.cpp, namely CreateVehicle() and UpdateStatus(). The CentralFunction() function code is shown below:

```
void CentralFunction()
{
        int instance;
        long val=2;

        instance = CreateVehicle();

        UpdateStatus(instance);
}
```

The CreateVehicle() function is responsible for CORSIM notifying the RTI that a vehicle has entered the network. The UpdateStatus() function serves to change the status attribute of each vehicle as it exits the on-ramp. A vehicle in the CORSIM federate has a status value of 1. This value is changed to 2 via the UpdateStatus () function upon the vehicle entering the other participating federate, namely the tollbooth exchange.

Finally, calls to RTI services for resigning and destroying the federation are in the upexit function located in upcntrl_HLA.cpp. These are used for clean up and release of the CORSIM federate from the federation. This function calls another function, ExitFunction(), which is in corsim_interface_RTI.cpp. The upexit function code is shown below:

```
void UPEXIT()
{
        //clean up
        //delete all objects that were created
        delete pNetwork;
        ExitFunction();
        CWRITE("You OK here",24);

}
```

Within ExitFunction(), calls are made to two other functions in traf_RTI.cpp, namely ResignFederation() and DestroyFederation(). The ExitFunction() function code is shown below:

```
void ExitFunction()
{
        ResignFederation();
        DestroyFederation();
}
```

The ResignFederation() function allows the CORSIM to withdraw from the network after it is done executing. The CORSIM federate simulates the feeder road for 900 seconds. The DestroyFederation() function gives the federate the ability to destroy the federation if it is the last executing federate. However, this will never be the case since the Arena federate runs for a much longer period of time.

4.3 Arena

Arena is a general purpose, flow oriented event simulation from the Rockwell Corporation. It has features that allow it to be used both as language and as a simulation

tool. It has a flexible and powerful object oriented design, capable of being used with different applications.

An important characteristic of Arena is that it allows users to create their own "user code" (routines) by editing appropriate files (userc.cpp in this case). The user needs to include two files, simlib.h and smsim.lib, when executing the user-written C/C++ source files. Simlib.h contains a complete set of SIMAN function prototypes, macros and typedefs to externally interface the code(C/C++ functions) with SIMAN. Smsim.lib is a library for SIMAN components. The functions described in userc.cpp (see Appendix 4) are called from SIMAN to allow linking in user-coded routines. In this federate, userc.cpp is accessed by using incorporating an EVENT block in the Arena model. These characteristics of Arena have proved to be quite helpful in applying the HLA to Arena, which is discussed in the following section.

4.3.1 Arena Federate

The Arena model that simulates a tollbooth exchange was created to test for compatibility with the HLA. This federate works in conjunction with the CORSIM model. Once again, this network has no signal control.

Figure 4.2 below depicts this simulation model.

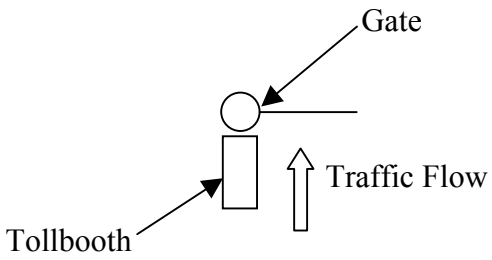


Figure 4.2. Arena Federate (Tollbooth Exchange)

Vehicles enter the tollbooth exchange after exiting the CORSIM federate via the on-ramp. All traffic is one-way. After vehicles are serviced at the tollbooth, the model disposes of them. The only objects defined in the simulation are vehicles with attributes of vehicle identification number, status, and CORSIM exit time. The change in the value of the status attribute of a vehicle triggers the process of the Arena federate creating an entity (vehicle) to go through the tollbooth exchange. This process is described in the next section.

4.3.2 Arena Interface Environment

Arena is interfaced with the RTI via the userc.cpp file existing in Arena. Userc.cpp consists of subroutines that can be customized and activated for use during a simulation run. In this instance, the cevent subroutine is used to generate the vehicles that will eventually go through the tollbooth exchange. The cevent subroutine is triggered using

an EVENT block in the Arena model. This construct, as well as the rest of the model, is shown below in Figure 4.3.

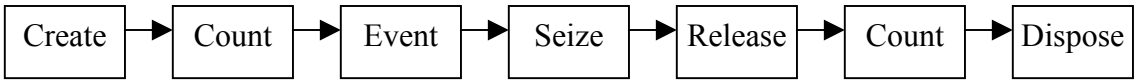


Figure 4.3. Arena Federate (Block Constructs)

The corresponding SIMAN code is shown below in Figure 4.4.

```
Model Frame
2$                  CREATE,      1,0.0:10,500:NEXT(3$);

3$                  COUNT:       total_count,1;
0$                  EVENT:       1;
Toll Booth Entry    SEIZE,       1,Other:
                                 Toll_Booth,1:NEXT(Toll Booth Exit);

Toll Booth Exit     RELEASE:     Toll_Booth,1;
1$                  COUNT:       toll_count,1;
4$                  DISPOSE:     No;

Experiment Frame
PROJECT,"External input test","Industrial Engineering",,,No,No,No,No,No,No,No;

RESOURCES:Toll_Booth,Capacity(1),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(N
o
,,);
COUNTERS:  total_count,,Replicate:
           toll_count,,Replicate;

REPLICATE,    1,,,Yes,Yes,,,,24,Seconds,No,No;
```

Figure 4.4. Model Frame and Experiment Frame for the Arena Federate

A project workspace file, named Msuserc.dsw, was created that contains the source and header file involved in interfacing with the RTI. These files are listed in Table 4.2 below.

Table 4.2

Source and Header Files in the Arena Interface Environment

Source Files	**Header Files**
arena_interface_RTI.cpp	FederateAmbassador.h
FederateAmbassador.cpp	Global_Variables.h
traf_RTI.cpp	simlib.h
userc.cpp	stdafx.h
	traf_RTI.h

Much like the CORSIM environment, most of the files already existed as part of the structure required for Arena. The files added to the structure to implement the HLA functionality are: arena_interface_RTI.cpp, FederateAmbassador.cpp, traf_RTI.cpp, FederateAmbasssador.h, Global_Variables.h, and traf_RTI.h. The following section details an execution of the Arena federate.

4.3.3 Arena Interface with the HLA

This section describes the interaction between the files listed in Table 4.2. A vehicle is followed throughout the simulation. The Arena federate, much like any participating federate, is capable of creating and/or joining a federation. When this federate is first

executed, the userc.cpp file is read and the logic is carried out. The section of the userc.cpp file relevant to this research effort is shown below.

```
void InitialFunction(void);
void ExitFunction(void);

//  the entry point within the model is "cevent"
//*********************************************************************
//
// Routine :      cevent
//
// Description : Maps the event number n to a call to the appropriate
//                event subroutine containing the logic for the event
//
// Inputs :       SMINT l - index of entity being processed by the event
//                SMINT n - event number
//
//
//*********************************************************************

long ticker=0;
extern long check;
extern RTI::RTIambassador    rtiAmb;

extern "C" void cdecl cevent  (SMINT entityLoc, SMINT evntNum)
{

        for (int k=1; k<3000; k++)
          rtiAmb.tick();
        switch(evntNum)   {
          case 1:
                if(check!=2)
                   dispos(&entityLoc);
                else
                        check=0;
        }
        ticker++;
        if(ticker==500)
                ExitFunction();

#ifdef TEST_FUNC
```

```
  sr_Printf  ("\n\nEntered cevent\n");
  sr_Printf  ("entityLoc    =%ld\n",   entityLoc);
  sr_Printf  ("evntNum    =%ld\n",   evntNum);
  sr_Printf  ("tnow  =%f\n",    gettnw());
  sr_Printf  ("numrun  =%ld\n\n", nrep());
#endif
}
```

The first function encountered is InitialFunction(). The InitialFunction() function code is shown below.

```
void InitialFunction()
{
        JoinFederation();
        GetAttributeHandles();
        PublishSubscribeAttributes();
}
```

This function resides in arena_interface_RTI.cpp (see Appendix 4) and makes calls to three other functions in traf_RTI.cpp (see Appendix 3), namely JoinFederation(), GetAttributeHandles(), and PublishSubscribeAttributes(). These functions perform the same way as in the CORSIM environment. The JoinFederation() function allows the Arena simulation to create a federation named “Feeder Road” or join a federation of the same name. Since the Arena simulation runs for a considerably longer time period than the CORSIM simulation, the Arena simulation is typically started first so that a greater number of vehicles pass through the network. This is also beneficial when observing the network to verify its proper functioning. The GetAttributeHandles() function gives the handles for the vehicles class and its attributes. The PublishSubscribeAttributes()

function permits the federate to show and capture information regarding each vehicle in terms of its attributes.

In order to simulate the passing of vehicles from the CORSIM federate to the Arena federate, the RTI, cevent subroutine in userc.cpp, and the EVENT block work in concert. As a vehicle passes the detector at the end of the on-ramp of the CORSIM federate, the CORSIM federate makes a call to the RTI to update the status attribute of the vehicle to a value of 2. This is done via the UpdateStatus() function in traf_RTI.cpp in the CORSIM environment workspace. The UpdateStatus() function code is shown below.

```
void UpdateStatus(int id)
{
        long val= 2;
        int tmp;

        tmp=htonl(val);

        for(int k=1; k<100; k++)
          rtiAmb.tick();

        RTI::AttributeHandleValuePairSet *pNameValuePairSet;

        pNameValuePairSet=RTI::AttributeSetFactory::create(1);

        pNameValuePairSet->add(statusID,(char*)&tmp,sizeof(tmp));

        try
        {
                rtiAmb.updateAttributeValues(RTI::ObjectHandle(id),

        *pNameValuePairSet,NULL);
                fout << "                Updated Status of vehicle" << endl;

        }
```

```
        catch (RTI::Exception& e)
    {
       fout << "Error [UpdateAnAttribute()]: " << &e << endl;
    }

    pNameValuePairSet->empty();

    delete pNameValuePairSet;
}
```

Since the Arena federate subscribes to changes in a vehicle's attributes, the change in the status attribute to a value of 2 is relayed to the Arena federate. This is achieved using a reflect attribute method. The corresponding code for this method in FederateAmbassador.cpp (see Appendix 5) is shown below.

```
void FedAmb::reflectAttributeValues(
   RTI::ObjectHandle                    theObject,
   const RTI::AttributeHandleValuePairSet& theAttributes,
   const char                           *theTag)
throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotKnown,
  RTI::FederateOwnsAttributes,
  RTI::FederateInternalError)
{

   RTI::AttributeHandle attrHandle;
   unsigned long valueLength;

   // Look up the object in the object table
         for (unsigned int i = 0; i < theAttributes.size(); i++) {
                  attrHandle = theAttributes.getHandle(i);
                  if (attrHandle == statusID) {
                           long value;

                           theAttributes.getValue(i, (char *)&value, valueLength);

                           check=ntohl(value);
                           ft<<"called"<<endl;
```

```
                }
        }
}
```

The change in the status attribute forces the Arena federate to release a vehicle into the tollbooth exchange. This is accomplished using the cevent subroutine in userc.cpp and the EVENT block. When the Arena federate is started, the CREATE block generates an entity that proceeds to the EVENT block. When this entity arrives at the EVENT block, control of the entity passes to the user-coded event (cevent subroutine). Within this subroutine, there is a check to see if the RTI has updated the status attribute of a vehicle to a value of 2. If there has been an update, this means a vehicle has exited the CORSIM federate via the on-ramp and is ready to proceed through the tollbooth exchange. Control then passes back to the EVENT block where the entity (vehicle) is released to the SEIZE block (tollbooth), processed, released, counted, and finally disposed.

This process continues until the CORSIM simulation completes execution after 900 time steps and resigns from the federation. The CORSIM federate then tries to destroy the federation but is unsuccessful since the Arena federate is continuing to run. The Arena federate continues to run until 30,000 entities have been created; far exceeding the simulation run time of the CORSIM federate. Being the last one to leave the simulation environment, the Arena federate executes the ExitFunction() function in userc.cpp, which deletes all objects and destroys the federation.

4.4 Results

A simple check was performed to determine if the two federates were interacting properly. At the end of the execution of both federates, the number of vehicles exiting the CORSIM federate at the on-ramp should equal the number of vehicles serviced by the tollbooth in the Arena federate. The number of vehicles that pass the detector at the end of the on-ramp is stored in a CORSIM variable and can be accessed by importing it into the application. The variable responsible for the cumulative number of vehicles passing the detector is DTMOD(DT). The number of vehicles serviced by the tollbooth in the Arena federate is counted using a COUNT block (toll_count) after the vehicle is released from the tollbooth. The resulting values obtained from DTMOD(DT) and toll_count were identical after several trial runs with each trial run using a different seed value. Table 4.3 below shows the results from 5 trial runs.

Table 4.3
Vehicle Counts from the CORSIM and Arena Federates

Trial #	DTMOD (DT) Value	Toll_count Value
1	393	393
2	394	394
3	393	393
4	394	394
5	393	393

The federation execution was run on two PCs operating on a local area network (LAN). The RTI was launched from a PC using Windows NT 4.0 as its operating system with dual Pentium Pro processors (200 MHz each) and 128 MB RAM. This PC also hosted

the CORSIM federate. The other PC used Windows NT 4.0 as its operating system with a Pentium III 400 MHz processor and 256 MB RAM. This PC hosted the Arena federate. The CORSIM simulation executes, on average, in 81.80 seconds; whereas the Arena simulation executes, on average, in 163.20 seconds. The CORSIM federate is simulating 15 minutes of traffic flow on the feeder road network while the Arena federate is simulating for a total of 30,000 entities being produced. A discussion on execution speed of the federation does not have much relevance in this situation. Most implementations of the HLA are on powerful workstations, not PCs. In our case, a PC was adequate and performed quickly due to the relatively small nature of the federation.

4.5 Conclusions

This chapter detailed the development of a distributed simulation environment that functions according to the HLA specification. The successful development of this environment demonstrates that it is feasible to make existing traffic simulation environments HLA compliant. HLA functionality can be implemented on existing traffic simulations as long as the traffic simulation software has an interface capability so it can communicate with the RTI. This was shown in the development of separate interfaces for the CORSIM and Arena federates, respectively. The next chapter will address the remaining research objectives.

CHAPTER V

HLA TOOLS AND PROCEDURES

5.1 Introduction

In Chapter III, an overview of the HLA was given. Necessary components for federates and the federation to be HLA compliant were identified. These components include: a simulation object model (SOM), a federation object model (FOM), and a Federation Execution Data (FED) file. The creation of these components has been greatly simplified with the development of a software tool by the DoD's Defense Modeling and Simulation Office (DMSO). The Object Model Development Tool (OMDT) provides a Windows-based environment in which to create the necessary components. This chapter discusses the use of the OMDT, procedures put in place by the DoD to test federates for HLA compliance, and web-based simulation with respect to the HLA. More specifically, this chapter discusses the accomplishment of the last three objectives of this research effort: (2) evaluate the usability of existing HLA support software in the transportation arena, (3) determine the usability of methods developed by the military to test for HLA compliance on traffic simulation models, and (4) examine the possibility of using the HLA to create Internet-based virtual environments for transportation research. The discussion will be relative to the simulation environment described in Chapter IV.

5.2 Object Model Development Tool

In order for a federation to be HLA compliant, each participating federate must have a SOM, and the federation, as a whole, must have a FOM. Since the CORSIM and Arena federates have the same object instances and attributes, the SOMs and the FOM are identical. This simplifies the creation of the object models in that the OMDT is used only once. The object models are essentially organized documentation of each federate and the federation. A series of tables must be completed so that the entire simulation environment is defined. This is done so that federation developers can easily determine if other federates (those created by someone else) are pertinent to their federation. If so, these "foreign" federates may be included in their federation execution.

HLA object models specify information about class of objects, their associations, attributes, and interactions. Current HLA framework requires that this information be in the form of tables. Federations and/or federates are not required to use all the tables, but only those that are relevant to their purpose. All object models must include an identification table, or "point of contact" table. This identifies the federate maker and facilitates the exchange of information between federation developers. For the federation and federates in this research effort only tables describing the class structure and attributes are necessary. There are no interactions between the vehicles and no vehicle parameters were established. This was intentionally done since the objective of the research was to simply demonstrate the ability for information exchange between two simulations using the HLA.

The contents of the identification, class structure, and attributes tables are shown in Tables 5.1 through 5.3 below.

Table 5.1

Object Model Identification Table

Category	Information
Name	Feeder Road
Version	1.0
Date	8/19/02
Purpose	Traffic simulation network
Application Domain	Transportation
Sponsor	Texas A&M University, College Station
POC (Title, First, Last)	Dr. Sharif Melouk
POC Organization	Department of Industrial Engineering
POC Telephone	979-845-5199
POC Email	sharif@tamu.edu

Table 5.2

Object Class Structure Table

Class1	Class2
Vehicle (P)	

Table 5.3

Attribute Table

Object	Attribute	Datatype	Cardinality	Units	Resolution	Accuracy
Vehicle	ID	Long	1			perfect
	Status	Short	1			perfect
	CorsimExitTime	Float	1	seconds		perfect

Accuracy Condition	Update Type	Update Condition	Transferable/Acceptable
always	Conditional		N
always	Conditional		N
always	Conditional		N

Updateable/Reflectable	Routing Space	Delivery	Message Ordering
U	N/A	best_effort	receive
U	N/A	best_effort	receive
U	N/A	best_effort	receive

Table 5.2 shows the class structure table. The only object class defined in the entire federation is the vehicle class. The "(P)" in the table indicates that vehicles are publishable. The second column, "Class2", and any other subsequent columns would be used if there were subclasses within the federation. For example, possible subclasses for the vehicle class would be cars, trucks, and buses. Table 5.3 shows the attribute table. The attributes of the vehicle class are listed. Once again, the attributes are ID, Status, and CorsimExitTime. The remainder of the table defines the properties of each attribute. The most pertinent properties to the federation under consideration are datatype, units, and updateable/reflectable. Lastly, sets of tables called lexicons are included in the

object models. Lexicons are used to explicitly define the classes, interactions, attributes, and parameters. For this federation, class and attribute lexicons are relevant. These lexicons are shown in Tables 5.4 and 5.5 below.

Table 5.4

Object Class Definitions (Class Lexicon)

Term	Definition
Vehicle	Defines the sole objects in the simulations

Table 5.5

Attribute Definitions (Attribute Lexicon)

Class	Term	Definition
Vehicle	ID	Identifies the vehicle
	Status	Status identifying the location of the object
	CorsimExitTime	Time vehicle exits from CORSIM simulation

5.3 Federate Compliance Testing

A federate compliance test process has been established by the DMSO to help the modeling and simulation community determine if their simulations are HLA compliant. The test is a four-step process that "ensures that a federate performs in accordance with the Interface Specification and the Object Model Template (OMT) standards, per the HLA compliance checklist" (DMSO, 1999). A series of tests are performed on the federate to determine if the federate is compliant. The CORSIM federate was subjected

to this testing process. The four-step process, with respect to this federate, is detailed below.

Step 1: Application

A request is made by the federate developer for an HLA Federate Compliance Test by completing a test application at the DMSO website, http://hlatest.msiac.dmso.mil/HLATest_1_3/htdocs/step1.html. The application is simply providing contact information for the federate developer and a description of the federate under test (FUT). Upon receiving the application, a Federate Certification Agent will determine the priority of this compliance test request. The request for a compliance test for the CORSIM federate was approved within two days of the initial application. Additionally, a test-ID number was assigned, and a user ID number and password are provided for access to the remaining parts of the test website necessary to complete the compliance test.

Step 2: Federate Conformance Notebook

The federate developer submits a Federate Conformance Notebook that consists of the SOM, the Federate Conformance Statement (CS), and (optional) Scenario Data. The CS lists the services that the federate promises to perform during the test process. The SOM (traffsim.omt), created using the OMDT, and the CS (FeederRoad.cs) were uploaded to the DMSO website.

The contents of the SOM were shown in the previous section, whereas the CS is shown in Table 5.6 below.

Table 5.6

CORSIM Federate Conformance Statement

SERVICE GROUP	SERVICE	IF Ref	OMT Ref	Check List	M/O
Create/Destroy	Create Federation Execution	4.2	None	Item 6	M
	Destroy Federation Execution	4.3	None	Item 6	M
Join/Resign	Join Federation Execution	4.4	None	Item 6	M
	Resign Federation Execution	4.5	None	Item 6	M
Publication and Subscription	Publish Object Class	5.2	4.2	Item 2	O
	Subscribe Object Class Attributes	5.6	4.2.2	Item 2	O
Object	Register Object Instance	6.2	4.2.2	Item 2	O
Representation	Update Attribute Values	6.4	4.4.2	Item 2	O
	Reflect Attribute Values †	6.5	4.4.2	Item 2	O

The certification agent checks the SOM for conformance to the OMT and then checks the SOM for consistency with the CS.

Step 3: Test Environment

The Certification Agent supplies the Federate Developer with the test sequence that will be performed in Step 4 of the compliance test process. Additionally, the Federate Developer provides information regarding the test environment. Test environment data includes: API used, federation execution host information, operating system, and hardware information. Table 5.7 below shows the information for the test environment of the CORSIM federate.

Table 5.7

Test Environment Information for CORSIM Federate

Category	Information
API Used	C++
Operating System	Microsoft Windows NT 4.0
RTI Execution Host Name	shannon-grad
RTI Execution IP Number	165.91.246.90
RTI Execution Port Number	5996
FED Execution Host Name	shannon-grad
FED Execution IP Number	165.91.246.90
Firewall	Yes
FED File	traffsim.fed
RID File	RTI.rid

The last two entries in the table (the FED and RID files) are configuration files required for proper functioning of the RTI. The FED file (see Appendix 6) contains the listing of object and interaction classes used by the federation and the RTI. It is used during the creation of the federation phase (DMSO, 1999). The FED file is automatically generated

from the FOM using the OMDT. The RID, RTI Initialization Data, file (see Appendix 7) contains configuration parameters that are used to control the operation of the RTI software (DMSO 1999). All parameters within this file have a default setting but can be customized to fit a particular federation execution. For example, for this testing process, in the Parameter Definition section of the RID file, the network address used by the RTI Executive process was edited to "shannon-grad:5996". This is a change from the default IP address of 224.9.9.2:22605. In addition to port 5996 being opened on the host computer, all high TCP ports (all above 1024) were opened temporarily during execution of the test sequence.

Step 4: Interface Test and Reporting

The federate developer and the Certification Agent execute the interface test. The interface test has two parts: the Nominal Test and the Representative SOM (RepSOM) test. The Nominal Test ensures that the FUT can invoke and respond to all services listed in its CS. The RepSOM test ensures that the FUT is capable of invoking and responding to services using a range of data contained in its SOM (DMSO, 2002). After the test is completed, the Certification Agent issues a Certification Summary Report that contains the results of the test.

5.4 Web-Based Simulation and the HLA

An area of growing interest in the simulation community is web-based simulation. There has been discussion on which technology or simulation language should be used

in developing web-based environments. The HLA has been included as a potential candidate to become the standard with respect to web-based simulation.

The driving force behind web-based simulation is the desire for simulation creators to collaborate in a distributed simulation-like setting. The main idea is to promote reuse and interoperability among simulations. These are the same reasons why the HLA was developed. So, the HLA seems a logical choice to become the web-based simulation standard. Technologically speaking, there are no barriers to prevent the HLA from being implemented on the web. A web-based environment could exist in which transportation federate developers logon at a website. Each developer would then execute their federate and begin communicating with the other federates via the RTI that is running on the host website's server. This type of situation also takes advantage of the HLA being non-software specific. Furthermore, this environment resembles a plug and play situation in which several different traffic simulations can be exchanged to create multiple transportation simulation environments to be analyzed.

5.5 Conclusions

This chapter detailed the development of the object models and the procedure to test for HLA compliance of simulation models. Additionally, there is a discussion on web-based simulation with respect to the HLA.

Existing HLA support software, such as the OMDT, proved to be very useful in transforming stand-alone traffic simulation models into HLA compliant traffic simulation models. In order to create the object models, each traffic simulation model must be broken down into its individual components such as objects and object attributes. The Windows-based layout of the OMDT allows for easy documenting of each component, thus creating the SOMs and FOM.

The compliance test process developed by the DoD is very beneficial and useful in ensuring that a newly developed traffic federate is HLA compliant. The organized structure of the step-by-step process ensures that the FUT delivers its stated functionality. Federate developers communicate directly with the certification agent, which is convenient if problems with testing arise. While the compliance test process ensures that a federate is HLA compliant, it does not validate the federate. In other words, the test does not guarantee that the federate is an accurate representation of the real world environment, only that the federate is able to communicate with other federates in accordance with the HLA specification. The CORSIM federate successfully completed the four-step compliance test process. The entire process, commencing with the test request and ending with the interface test, lasted 40 days. An official Certification Letter and a Certificate of HLA Compliance for Feeder Road was issued. The compliance certificate is shown in Appendix 8.

The HLA has great promise in becoming the technology standard in creating Internet-based virtual transportation environments. The benefits of the HLA, such as simulation reuse and interoperability, align with the web-based simulation goals of increased collaboration among simulationists and reuse of simulation models. HLA-driven virtual transportation environments may lead to more cooperative efforts between national and state transportation agencies as well as private transportation consultants due to the ease of model sharing resulting from the HLA implementation.

CHAPTER VI

CONCLUSION AND FUTURE RESEARCH

This dissertation deals with the High Level Architecture (HLA) and the accomplishment of four research objectives. These research objectives are: (1) determine the feasibility of making existing traffic management simulation environments HLA compliant; (2) evaluate the usability of existing HLA support software in the transportation arena; (3) determine the usability of methods developed by the military to test for HLA compliance on traffic simulation models; and (4) examine the possibility of using the HLA to create Internet-based virtual environments for transportation research. The HLA is an object-oriented approach to distributed simulations developed by the Department of Defense (DoD) under the Defense Modeling and Simulation Office (DMSO). It is intended to handle the issues of reuse and interoperability of simulations. Although the HLA was developed for use in military applications, it has great potential for civilian applications including transportation systems modeling.

The first objective was accomplished by creating a distributed traffic environment to explore and demonstrate the usefulness of the HLA and consisted of two traffic simulations developed using two different software packages, namely CORSIM and Arena. CORSIM is a microscopic simulation software developed by the Federal Highway Administration. It models surface streets and freeways along with a wide array of traffic control devices. The CORSIM model, or federate, used in this study simulates

a freeway feeder road with a freeway on-ramp with no signal control. Arena is a widely known, general purpose simulation software. It is capable of modeling many different types of environments, including a traffic environment. The Arena federate models a freeway tollbooth exchange where the entering vehicles come from the vehicles leaving the freeway on-ramp of the CORSIM federate. The HLA functionality was successfully implemented upon the two separate models, therefore allowing the federates to communicate with one another under the HLA concept of federation.

In addition to the basic requirements of a distributed simulation such as synchronization and data exchange, each federate must satisfy a technical requirement in order to interoperate in an HLA federation, that being the ability to exchange information with the RTI using the dual ambassador principle. This requires each federate to construct proper calls to the methods of the RTI ambassador object and provide a federate ambassador object, which contains callback functions, which in turn can be called by the RTI. While each of these requirements can be easily satisfied for a simulation written in C++, it can be a somewhat more difficult task to satisfy them for a simulation written using other languages/tools. Straβburger (1999) suggested four possible methods for implementing the HLA functionality into existing simulation tools.

These methods are listed below.

1. Re-implementation of the tool with HLA extensions

This is possible if the source code of the tool is available. It is the simplest and most desirable solution, as it would eliminate the need to program. However, for most commercially available simulation tools, this can be made possible only by the tool developer. Several software companies are considering such additions if the HLA increases in popularity.

2. Extension of intermediate code

Some simulation tools translate model descriptions written in a tool-dependent modeling language into another programming language (e.g. C++). It is possible to modify this code to realize the HLA extensions. Since this code is compiler generated, typically it is not an easy solution to the problem, and an automated solution is desirable.

3. External programming interface

This is possible if the tool offers an extensible and open architecture. The tools should offer a library interface (a DLL interface in Windows) with the ability to call arbitrary functions or methods in these libraries.

4. Coupling via a gateway program

The last solution for tools that cannot be connected to the RTI by any of the prior methods is the development of a gateway program. The gateway program could communicate with the simulation tool via appropriate means (e.g. files, pipes, ports, network) depending on the capabilities of the simulation tool.

An interface was created for both Arena and CORSIM, using method 3 from above, allowing each federate to communicate with the Run-Time Infrastructure (RTI). The RTI is a type of "operating system" that provides HLA services to the participating federates of the simulation environment (federation). C++ was the interface language used to connect each federate to the RTI. In CORSIM, the run-time extension (RTE) feature was used. The RTE allows externally written code to be incorporated in the simulation during its execution. Several source and header files were created to interact with existing code in CORSIM and the RTI, thus linking the software. Similarly, in Arena, a user interface file, userc.cpp, was used for interfacing. This file, already existing within the Arena software, consists of several subroutines that can be called and executed to perform special procedures during federation execution. This file, along with other added files, interacts with the block and element constructs in Arena and the RTI, thus linking the software. In general, this federation execution is essentially an exercise in transferring the objects (vehicles) in the CORSIM federate to the Arena federate.

The HLA functionality was successfully implemented on the CORSIM and Arena federates. This accomplishment emphasizes the point that the HLA is not software specific. The reason for selecting these two software packages is two-fold: both packages are very popular in the simulation community, and both have an interface capability. While CORSIM trails Trafficware's SimTraffic transportation software in terms of use by the transportation simulation community, CORSIM does have an interface capability, whereas SimTraffic does not. Thus, it is clear that not all commercial simulation packages lend themselves for use in HLA applications. An interface capability is absolutely necessary so that the simulation (federate) can communicate with the RTI to take advantage of the HLA services. Otherwise, the actual source code of the simulation software must be altered to incorporate the HLA concepts. This is often an unrealistic option since most federate developers do not have access to the software's source code. Moreover, in the event the code is accessible, the prospect of essentially developing new software is overwhelming, not to mention very time consuming and rather costly. In the case of Arena, it is a very commonly used

simulation package and is known by practically everyone in the simulation community. In addition, Arena's interface feature is easy to access, and the software can model a multitude of environments, including transportation.

During the course of this investigation, several commercial simulation packages were identified and evaluated for HLA interface feasibility. If the HLA interface seems feasible, the most appropriate interface method (from the four methods suggested above) has been identified, depending upon the functionalities offered by the tool. The suggestions offered are based on our and other researchers experience of developing prototype HLA federations using some of these tools. Tools for which there is no HLA interface example available, the suggestions are based on the technical capabilities of the tool and the information given by the vendor.

Table 6.1 below lists the simulation packages evaluated.

Table 6.1

Simulation Package Evaluation

No.	Software	HLA Interface Feasible	Prototype Available	Interface Method	Vendor
1	AnyLogic	Yes	No	External Interface	XJ Technologies
2	Arena	Yes	Yes	External Interface	Rockwell Software
3	AutoMod	Yes	Yes	External Interface	Brooks-PRI Automation
4	Awe Sim	Yes	No	External Interface	Frontstep Inc.
5	Decision Pro	No	No	Possibly by using a Gateway program	Vanguard Software Corporation
6	Extend Suite	Yes	No	External Interface	Imagine That Inc.
7	Flexsim ED	Yes	No	External Interface	FlexSim Software Products
8	GAUSS	Yes	Yes	External Interface	Aptech Systems
9	GPSS/H	Yes	Yes	Extension of intermediate Code for JavaGPSS Coupling via gateway program for GPSS/H	Wolverine Software Corporation
10	MAST	No	No	Possibly by using a Gateway program	CMS Research
11	ProModel	Yes	Yes	External Interface	ProModel Corporation
12	Quest	Yes	No	External Interface	Delmia Corporation
13	RDK	Yes	No	External Interface	Palisade Corporation
14	SIGMA	Yes	No	Extension of Intermediate Code	Custom Simulation
15	SimScript II.5	Yes	No	External Interface	CACI International
16	SIMUL8	Yes	Yes	External Interface	Simul8 Corporation
17	SLX	Yes	Yes	External Interface	Wolverine Software Corporation
18	Witness	No	No	Possibly via a gateway program	Lanner Group

Research objectives (2) and (3) were accomplished by evaluating tools and procedures developed by the DMSO to aid in making military simulations HLA compliant. These

tools and procedures can also be used on non-military simulations. The creation of the object models and some of the configuration files can be rather tedious. However, the DMSO's Object Model Development Tool (OMDT) provides a windows-based software environment in which these can be easily created. The DMSO has also established a four-step HLA compliance test process that a federate must successfully complete so that it can be deemed HLA compliant. A Certification Agent from the DMSO administers this test via the Internet. The test ensures that the simulation (federate) performs in accordance with the Interface Specification and the Object Model Template standards. In other words, the federate must perform the services that are detailed in its SOM.

Organizations should consider adopting the HLA as their permanent simulation architecture. This is especially true in situations where an organization depends heavily on simulations and the connection of them in a distributed setting. A heavy reliance on simulation to analyze system performance most likely translates to a significant amount of time spent on maintaining and adapting the simulation models. Implementing the HLA on the models would reduce the adaptation time and aid in the analysis of multiple environments. Depending on the volume of simulation models produced by said organization, a separate group within the organization (much like the DoD) could be established to determine the compliance of all simulation models with the HLA specification. In cases where few models are developed, a separate testing group is not

necessary, so the individual federate developers would be responsible for compliance verification of each model.

As research objective (4) states, the HLA was examined to determine its viability in creating Internet-based virtual transportation environments. From a technical standpoint, there seems to be no barriers preventing the HLA from becoming the technology standard that drives web-based transportation simulation environments. The HLA promotes collaboration among simulation creators since the transportation simulations participating in the web-based environment will be HLA compliant, and hence, have reuse and interoperability properties. Plug and play type environments will result and lead to the analysis of multiple and diverse transportation systems.

While the HLA has tremendous potential to become a simulation standard, technical and economic barriers could prevent this from happening. The two main economic barriers are (1) implementing the HLA concept and (2) the cost of necessary computer hardware, software, and network capabilities. The cost of implementation far exceeds the cost of ensuring computer and network capabilities. Implementation costs include the cost of salaries and training for simulation creators and supporting staff, and the transformation of existing simulations to HLA-compliant simulations. For example, HLA training classes are available through privately held engineering and software development consultant companies. Distributed Simulation Technology, Inc. offers a 4-day class on the HLA for $1795 per person at their facility in Florida. Additionally, the DMSO and

the McLeod Institute of Simulation Sciences at California State University, Chico are cooperating in developing the HLA University Outreach Program. This program is disseminating information about the HLA to colleges and universities at the undergraduate, graduate, and research levels. The program goals are to inform simulation users at universities of the applicability of the HLA to a broad range of simulation problems, to encourage application of the HLA to non-DoD models, to stimulate research into open HLA-related problems, and to educate students in the use of the HLA as a valuable job skill.

While the initial cost of transitioning to HLA-compliant models may be significant, subsequent costs will be far less since the HLA promotes the reusability and interoperability of simulation models. Furthermore, since the implementation of the HLA functionality does not require specific software, costs will be reduced with respect to purchasing computer software and software licenses. This should encourage more organizations to experiment with the HLA as their simulation standard.

Determining the exact overall cost of implementing the HLA is quite difficult since several variables can come into play. However, looking at the additional cost on an individual simulation model basis, implementing the HLA should not increase initial model development costs by more than 10-15%. A range is given because the implementation of the HLA is directly related to the number of features in the simulation model that must be made HLA compliant; features desired to be made available to the

other federates. As the number of features increases, the time and cost of implementation will increase accordingly. Of course, this estimate assumes that the simulation creator is well versed with the simulation software and the concepts of the HLA.

Another potential barrier to widespread adoption of the HLA as a simulation standard is the unwillingness of the simulation community to change. Often, simulation creators become complacent and satisfied with the status quo. However, if the benefits and advantages of the HLA discussed above are relayed to the community as a whole, the technology may become commonplace. This would be a great benefit to transportation simulation creators and analysts. Situations could arise in which organizations, cities, and possibly states work jointly to alleviate current transportation problems and develop new transportation management strategies. The sharing of transportation management models and ideas would be greatly enhanced by the HLA due to the structured nature and documentation requirements of the technology.

Future research could explore opportunities with implementing the HLA with other simulation packages. There are many types of simulation software currently being used. There are many potential application areas in which linking two or more independent simulations, each created in a different simulation package, would greatly benefit a company or industry in terms of analyzing a process, method, or procedure. The HLA would be a viable solution for linking the simulations given that each of the simulation

packages involved have some sort of interfacing capabilities similar to CORSIM and/or Arena. As long as the software has the ability to call functions in the RTI, the HLA is a possible solution for the distributed environment. The possibilities with the HLA seem limitless in terms of the type of environment that it can be applied. Other possible industries, or environments, include, but are not limited to, manufacturing, logistics, and service. Finally, the area of Homeland Security has become very important and widely discussed. To fully address the issue of Homeland Security, many agencies and organizations must cooperate to achieve established goals. A useful tool for studying and examining possible security strategies is distributed simulation. Certainly, many simulations already exist that would be of even more benefit by linking them together. A likely vehicle for this linkage is the HLA. This would enable many security strategies to be examined in relatively short periods of time.

REFERENCES

S. Bachinsky, R. Briggs, B. Gibson and C. Turrell, "How to Plan, Set-up, and Run a Distributed Simulation Exercise," presented at Simulation Interoperability Workshop, Orlando, FL, September 2000.

A. Buss and L. Jackson, "Distributed Simulation Modeling: A Comparison of HLA, CORBA, and RMI," presented at Winter Simulation Conference, Washington, D.C., December 1998.

DMSO, "HLA Hands-On Practicum Course Material," August 2000.

DMSO, "RTI 1.3-Next Generation Programmer's Guide," October 1999.

M. M. Horst, D. Rosenbaum, K. A. Crawford and M. B. Woldt, "Improvements to the HLA Federate Compliance Testing Process," presented at Simulation Interoperability Workshop, Orlando, FL, September 2000.

M. M. Horst and M. B. Woldt, "Using HLA Tools to Facilitate Compliance Testing," presented at Simulation Interoperability Workshop, Orlando, FL, March 2000.

U. Klein, "Simulation-based Distributed Systems: Serving Multiple Purposes through Composition of Components", *Safety Science* **35**, 29-39 (2000).

U. Klein, T. Schulze and S. Straβburger, "Traffic Simulation Based on the High Level Architecture," presented at Winter Simulation Conference, Washington, D.C., December 1998.

J. Kuljis and R. J. Paul, "A Review of Web Based Simulation: Whither we wander?," presented at Winter Simulation Conference, Orlando, FL, December 2000.

T. McLean and R. Fujimoto, "The Federated-Simulation Development Kit (FDK): A "Source-Available" HLA RTI," presented at Simulation Interoperability Workshop, Orlando, FL, March 2001.

J.A. Miller, P.A. Fishwick, S.J.E. Taylor, P. Benjamin and B. Szymanski, "Research and Commercial Opportunities in Web-Based Simulation", *Simulation Practice and Theory* **9**, 55-72 (2001).

A. Ozaki, M. Furuichi, N. Nishi and E. Kuroda, "The Use of High Level Architecture in Car Traffic Simulations", *Transactions on Information and Systems* **10**, 1851-1859 (2000).

D. Pace, “Simulation Conceptual Model Role in Determining Compatibility of Candidate Simulations for a HLA Federation,” presented at Simulation Interoperability Workshop, Orlando, FL, March 2001.

E. Page, “The Rise of Web-based Simulation: Implications for the High Level Architecture,” presented at Winter Simulation Conference, Washington, D.C. December 1998.

E. Page and J. Opper, “Investigating the Application of Web-Based Simulation Principles within the Architecture for a Next-Generation Computer Generated Forces Model”, *Future Generation Computer Systems* **17**, 159-169 (2000).

D. J. Paterson, E. S. Hougland and J. J. Sanmiguel, “A Gateway/Middleware HLA Implementation and the Extra Services that can be provided to the Simulation,” presented at Simulation Interoperability Workshop, Orlando, FL, September 2000.

S.M. Pratt and T.D. Dugone, “DIS to HLA Compliance in 21 days”, TRADOC Analysis Center, Monterey, CA, 1999.

C. Rouget and P. Henry, “Utilising HLA Object Models within a C++ Repository to generate tools to support the Federation Development and Execution Process,” presented at Simulation Interoperability Workshop, Orlando, FL, March 2000.

G. Sauerborn, G. Moss, G. Tan, F. Moradi, R. Ayani and P. Oxenberg, “HLA Ownership Management Services: We Almost Got It Right,” presented at Simulation Interoperability Workshop, Orlando, FL, September 2000.

T. Schulze, S. Straβburger and U. Klein, “Migration of HLA into Civil Domains: Solutions and Prototypes for Transportation Applications”, Simulation **73**(5): 296-303 (1999).

S. Straβburger, “On the HLA-based Coupling of Simulation Tools”, Institute for Simulation and Graphics, Department of Computer Science, Otto-von-Guericke University, Magdeburg, Germany, 1999.

S. Straβburger, T. Schulze, U. Klein and J. Henriksen, “Internet-Based Simulation using Off-The-Shelf Simulation Tools and HLA,” presented at Winter Simulation Conference, Washington, D.C., December 1998.

M. R. Stytz and S. B. Banks, “Enhancing the Design and Documentation of High Level Architecture Simulation Using the Unified Modeling Language,” presented at Simulation Interoperability Workshop, Orlando, FL, March 2001.

APPENDICES

APPENDIX 1

```
#include "netsim.h"
#include "network.h"
#include "upcntrl.h"
#include "stdlib.h"
#include "stdio.h"
#include "link.h"
#include "conio.h"
#include "node.h"
#include "detector.h"
#include <vector>

using namespace std;

extern "C" { void UPINIT(char *fname); }
extern "C" { void UPCNTRL(void); }
extern "C" { void UPEXIT(void); }
extern "C" {void __stdcall CWRITE (char *a, unsigned int length_arg);};

void InitialFunction(void);
void ExitFunction(void);
void CentralFunction(void);

void UPINIT( char *fname )
{
        //initialization routine
        //called once at the beginning of simulation
        int i;
        int j;


        char outbuffer[132];
        InitialFunction();

        sprintf(outbuffer,"init done \n");
//      CWRITE(outbuffer,132);

        endOfInit=0;
        prevInit=0;

        pNetwork=new CNetwork();
```

```
	//fname must be null terminated
	for (i=2;i<512;i++)
	{
		if (((fname[i-2]=='T')||(fname[i-2]=='t'))&&
			((fname[i-1]=='R')||(fname[i-1]=='r'))&&
			((fname[i]=='F')||(fname[i]=='f')))
		{
			j=i+1;
		}
	}

	for (i=j;i<512;i++)
	{
		fname[i]='\0';
	}

	pNetwork->m_TrafInputFile=CString(fname);

	//read the traf file
	pNetwork->ReadTrafFile();
	return;
}

void UPCNTRL()
{
	int time;
	BOOL init;
	POSITION pos, detpos;
	CLink* pLink;
	CDetector* pDetector;
	static vector<int>count;

	init=yinit;

	//the algorithm that controlls the signal states at the
	//intersections assumes time is always increasing, but
	//the CORSIM clock starts over after initialization
	//so the time at which initialization is over must
	//be recorded
	if ((!init)&&(prevInit))
	{
		//end of initialization
		endOfInit=prevTime+1;
```

```
	}

	//adjust the time by adding the end of initialization
	time=sclock+endOfInit;

	//get signal state for the node under corsim control
	//pNetwork->UpdateNodeSignalStates();

	//process any detector information

	pos=pNetwork->m_LinkList.GetHeadPosition();
	while (pos!=NULL)
	{
		pLink=pNetwork->m_LinkList.GetNext(pos);

		if(pLink->m_upnode->m_id==1 && pLink->m_dnnode->m_id==2)
		{
			pLink->ProcessDetectors();
			detpos = pLink->m_listOfDetectors.GetHeadPosition();
			pDetector = pLink->m_listOfDetectors.GetNext(detpos);
			count.push_back(pDetector->m_count);
			if(count[count.size()-1]-count[count.size()-2] && time>300) {
/*
				char outbuffer[132];
				char ch[20];
				_itoa (time-300,ch,10);
				sprintf(outbuffer,ch);
				CWRITE(outbuffer,132);
*/
				CentralFunction();
			}
		}

	}
	//record whether the simulation has reached equilibrium
	//or not, so the time at which initialization can be
	//recorded
	prevInit=init;
	prevTime=time;

}

void UPEXIT()
{
```

```
        //clean up
        //delete all objects that were created
        delete pNetwork;
        ExitFunction();
        CWRITE("You OK here",24);

}
```

APPENDIX 2

```
#include "traf_RTI.h"

void InitialFunction()
{
	JoinFederation();
	GetAttributeHandles();
	PublishSubscribeAttributes();
}

void ExitFunction()
{
	ResignFederation();
	DestroyFederation();
}

void CentralFunction()
{
	int instance;
	long val=2;

	instance = CreateVehicle();

	UpdateStatus(instance);
}
```

APPENDIX 3

```
#include <stdio.h>
#include <sys/types.h>
#include <fstream>

#ifdef _WIN32
        #include <winsock2.h>
   #include <process.h>
   #include <windows.h>
   #define getpid _getpid
#else
   #include <unistd.h>
   #include <netinet/in.h>
#endif

#include "traf_RTI.h"
#include "FederateAmbassador.h"
#include "Global_Variables.h"

void SleepSeconds(int howlong)
{
#ifndef _WIN32
   sleep(howlong);
#else
   Sleep(howlong*1000);
#endif
}

ofstream fout("log.txt");

RTI::RTIambassador    rtiAmb;      // RTI Ambassador
FedAmb                fedAmb;      // Federate Ambassador


void JoinFederation(void)
{
 char federateName[100];
 char hostName[256];
 int ready=0;

#ifdef _WIN32
```

```
        WSADATA wsaData;
   int err;

   err = WSAStartup( MAKEWORD( 2, 2 ), &wsaData );
   if ( err != 0 )
      sprintf(hostName, "UNKNOWN");
   else
      gethostname(hostName, 256);
   WSACleanup();
#else
   gethostname(hostName, 256);
#endif

  sprintf(federateName, "%s_%s-%s",federationName, hostName, "Feeder Road");

  try
  {
   fout <<"Invoking createFederationExecution"<<endl;
        rtiAmb.createFederationExecution(federationName,fedfileName);
   fout << federationName << " federation created"<< endl;
  }
  catch(RTI::FederationExecutionAlreadyExists& e)
  {
   fout << " Federation "<<federationName<<" already exists" << endl;
  }
  catch(RTI::Exception& e)
  {
   fout << "   EXCEPTION: caught in createFederationExecution call" << endl;
   fout << "   " << & e << endl;
   fout << "   TERMINATING" << endl;
   exit(1);
  }

  // Try to join the federation.  If it doesn't exist, wait a while and
  // try again.
  while (!ready)
  {
     ready = 1;

     try
     {
       rtiAmb.joinFederationExecution(federateName, federationName, &fedAmb);
                  fout<< "          Created federate "<<federateName << endl;
```

```
    }

    catch (RTI::FederationExecutionDoesNotExist& e)
    {
     // If the federation does not exist, then sleep for a second
     // and try to join again

                cerr << "Waiting for federation execution: " << &e << endl;
     ready = 0;
     SleepSeconds(1);
    }
    catch(RTI::Exception& e)
          {
                fout << "   EXCEPTION: caught in joinFederationExecution call" <<
endl;
                fout << "   " << & e << endl;
                fout << "   TERMINATING" << endl;
                exit(1);
          }

   }
 return;
}

void GetAttributeHandles(void)
{

        try {

          objectRootClassID = rtiAmb.getObjectClassHandle("objectRoot");
          privilegeToDeleteValueID =
        rtiAmb.getAttributeHandle("privilegeToDelete", objectRootClassID);

          //Handle for Vehcile class
          vehicleClassID = rtiAmb.getObjectClassHandle("Vehicle");

          //Handle for Attributes of vehicle class
          vinID = rtiAmb.getAttributeHandle("ID",vehicleClassID);
          statusID = rtiAmb.getAttributeHandle("Status",vehicleClassID);
          corsimExitTimeID =
rtiAmb.getAttributeHandle("CorsimExitTime",vehicleClassID);

          fout << "                Handles created successfully" << endl;
        }
```

```
        catch ( RTI::Exception& e )
    {
     cerr << "Error getting attributes: " << &e << endl;
          // Resign from "FoodFight" federation
          ResignFederation();
          rtiAmb.tick();
          DestroyFederation();
    }
}

void PublishSubscribeAttributes(void)
{
        RTI::AttributeHandleSet *attributes;

    attributes = RTI::AttributeHandleSetFactory::create(4);

    attributes->add(privilegeToDeleteValueID);
    attributes->add(vinID);
    attributes->add(statusID);
        attributes->add(corsimExitTimeID);

    try
    {
       rtiAmb.subscribeObjectClassAttributes(vehicleClassID, *attributes);

       rtiAmb.publishObjectClass(vehicleClassID, *attributes);

                  fout << "                Published and subscribed to all attributes
successfully" << endl;
    }
    catch (RTI::Exception& e)
    {
       cerr << "Error [PublishSubscribeAttribute()]: " << &e << endl;
    }
    attributes->empty();

    delete attributes;
}

void ResignFederation(void)
{

  try
```

```
  {
        // clear the callback queue prior to resigning
   for (int i=0;i<3;i++)
     rtiAmb.tick();


rtiAmb.resignFederationExecution(RTI::DELETE_OBJECTS_AND_RELEASE_ATTR
IBUTES);
  }
  catch(RTI::Exception& e)
  {
   fout << "   EXCEPTION: caught in resignFederationExecution call" << endl;
   fout << "   " << & e << endl;
   fout << "   TERMINATING" << endl;
   exit(1);
  }
}



void DestroyFederation(void)
{

  try
  {
   rtiAmb.destroyFederationExecution(federationName);
  }
  catch(RTI::Exception& e)
  {
   fout << "   EXCEPTION: caught in destroyFederationExecution call" << endl;
   fout << "   " << & e << endl;
        fout << " TERMINATING"<<endl;
        exit (1);
  }
}

int CreateVehicle(void)
{
        RTI::ObjectHandle id;

   try
   {
      id = rtiAmb.registerObjectInstance(vehicleClassID);
                fout << "                Created Vehicle with success" << endl;
```

```
   }
   catch (RTI::Exception& e)
   {
      fout << "Error [CreateWidget()]: " << &e << endl;
   }

        for(int k=1; k<100; k++)
          rtiAmb.tick();

        return (int)id;

}


void UpdateStatus(int id)
{
        long val= 2;
        int tmp;

        tmp=htonl(val);

        for(int k=1; k<100; k++)
          rtiAmb.tick();


        RTI::AttributeHandleValuePairSet *pNameValuePairSet;

        pNameValuePairSet=RTI::AttributeSetFactory::create(1);

        pNameValuePairSet->add(statusID,(char*)&tmp,sizeof(tmp));

        try
        {
                rtiAmb.updateAttributeValues(RTI::ObjectHandle(id),

        *pNameValuePairSet,NULL);
                fout << "                Updated Status of vehicle" << endl;

        }
        catch (RTI::Exception& e)
   {
      fout << "Error [UpdateAnAttribute()]: " << &e << endl;
   }
```

```
    pNameValuePairSet->empty();

    delete pNameValuePairSet;
}
```

APPENDIX 4

```
#include "traf_RTI.h"

void InitialFunction()
{
	JoinFederation();
	GetAttributeHandles();
	PublishSubscribeAttributes();
}

void ExitFunction()
{
	ResignFederation();
	DestroyFederation();
}
```

APPENDIX 5

```
#include <stdio.h>
#include <stdlib.h>
#include <fstream.h>


#include <sys/types.h>

#ifdef _WIN32
        #include <winsock2.h>
   #include <process.h>
   #include <windows.h>
   #define getpid _getpid
#else
   #include <unistd.h>
   #include <netinet/in.h>
#endif

#include "FederateAmbassador.h"

#include "traf_RTI.h"
long check=0;
ofstream ft("int.txt");

extern RTI::ObjectClassHandle vehicleClassID;
extern RTI::ObjectClassHandle objectRootClassID;
extern RTI::AttributeHandle   privilegeToDeleteValueID;
extern RTI::AttributeHandle vinID;
extern RTI::AttributeHandle statusID;
extern RTI::AttributeHandle corsimExitTimeID;


void FedAmb::startRegistrationForObjectClass(
   RTI::ObjectClassHandle        theClass)
throw (
  RTI::ObjectClassNotPublished,
  RTI::FederateInternalError)
{

}
```

```
void FedAmb::discoverObjectInstance(
   RTI::ObjectHandle          theObject,
   RTI::ObjectClassHandle     theObjectClass,
   const char                 *theTag)
throw (
  RTI::CouldNotDiscover,
  RTI::ObjectClassNotKnown,
  RTI::FederateInternalError)
{

   // Insert code here

}

void FedAmb::reflectAttributeValues(
   RTI::ObjectHandle                    theObject,
   const RTI::AttributeHandleValuePairSet& theAttributes,
   const char                           *theTag)
throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotKnown,
  RTI::FederateOwnsAttributes,
  RTI::FederateInternalError)
{
            // Insert code here


   RTI::AttributeHandle attrHandle;
   unsigned long valueLength;

   // Look up the object in the object table
         for (unsigned int i = 0; i < theAttributes.size(); i++) {
                  attrHandle = theAttributes.getHandle(i);
                  if (attrHandle == statusID) {
                           long value;

                           theAttributes.getValue(i, (char *)&value, valueLength);

                           check=ntohl(value);
                           ft<<"called"<<endl;
                  }
         }
```

```
}

void FedAmb::receiveInteraction (
      RTI::InteractionClassHandle       theInteraction,
  const RTI::ParameterHandleValuePairSet& theParameters,
  const char                        *theTag)
throw (
  RTI::InteractionClassNotKnown,
  RTI::InteractionParameterNotKnown,
  RTI::FederateInternalError)
{

   // Insert code here

}

void FedAmb::removeObjectInstance(
   RTI::ObjectHandle          theObject,
   const char                 *theReason)
throw (
  RTI::ObjectNotKnown,
  RTI::FederateInternalError)
{

   // Insert code here
}

void FedAmb::provideAttributeValueUpdate(
   RTI::ObjectHandle                theObject,
   const RTI::AttributeHandleSet& theAttributes)
throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotKnown,
  RTI::AttributeNotOwned,
  RTI::FederateInternalError)
{

}
```

APPENDIX 6

```
(FED)
(Federation FederationName)
(FEDversion v1.3)
 (spaces
 )
 (objects
   (class ObjectRoot
     (attribute privilegeToDelete  reliable timestamp)
     (class RTIprivate)
 (class Vehicle
    (attribute ID best_effort receive)
    (attribute Status best_effort receive)
    (attribute CorsimExitTime best_effort receive)
 )
 (class Manager
 (class Federation
    (attribute FederationName reliable receive)
    (attribute FederatesInFederation reliable receive)
    (attribute RTIversion reliable receive)
    (attribute FEDid reliable receive)
    (attribute LastSaveName reliable receive)
    (attribute LastSaveTime reliable receive)
    (attribute NextSaveName reliable receive)
    (attribute NextSaveTime reliable receive)
 )
 (class Federate
    (attribute FederateHandle reliable receive)
    (attribute FederateType reliable receive)
    (attribute FederateHost reliable receive)
    (attribute RTIversion reliable receive)
    (attribute FEDid reliable receive)
    (attribute TimeConstrained reliable receive)
    (attribute TimeRegulating reliable receive)
    (attribute AsynchronousDelivery reliable receive)
    (attribute FederateState reliable receive)
    (attribute TimeManagerState reliable receive)
    (attribute FederateTime reliable receive)
    (attribute Lookahead reliable receive)
    (attribute LBTS reliable receive)
    (attribute MinNextEventTime reliable receive)
```

```
    (attribute ROlength reliable receive)
    (attribute TSOlength reliable receive)
    (attribute ReflectionsReceived reliable receive)
    (attribute UpdatesSent reliable receive)
    (attribute InteractionsReceived reliable receive)
    (attribute InteractionsSent reliable receive)
    (attribute ObjectsOwned reliable receive)
    (attribute ObjectsUpdated reliable receive)
    (attribute ObjectsReflected reliable receive)
)
)
   )
)
 (interactions
   (class InteractionRoot  reliable timestamp
     (class RTIprivate  reliable timestamp)
  (class Manager reliable receive
  (class Federate reliable receive
    (parameter Federate)
   (class Request reliable receive
  (class RequestPublications reliable receive
 )

  (class RequestSubscriptions reliable receive
 )

  (class RequestObjectsOwned reliable receive
 )

  (class RequestObjectsUpdated reliable receive
 )

  (class RequestObjectsReflected reliable receive
 )

  (class RequestUpdatesSent reliable receive
 )

  (class RequestInteractionsSent reliable receive
 )

  (class RequestReflectionsReceived reliable receive
 )
```

```
 (class RequestInteractionsReceived reliable receive
)

 (class RequestObjectInformation reliable receive
   (parameter ObjectInstance)
 )

)

 (class Report reliable receive
 (class ReportObjectPublication reliable receive
   (parameter NumberOfClasses)
    (parameter ObjectClass)
    (parameter AttributeList)
 )

 (class ReportObjectSubscription reliable receive
   (parameter NumberOfClasses)
    (parameter ObjectClass)
    (parameter Active)
    (parameter AttributeList)
 )

 (class ReportInteractionPublication reliable receive
   (parameter InteractionClassList)
 )

 (class ReportInteractionSubscription reliable receive
   (parameter InteractionClassList)
 )

 (class ReportObjectsOwned reliable receive
   (parameter ObjectCounts)
 )

 (class ReportObjectsUpdated reliable receive
   (parameter ObjectCounts)
 )

 (class ReportObjectsReflected reliable receive
   (parameter ObjectCounts)
 )

 (class ReportUpdatesSent reliable receive
```

```
  (parameter TransportationType)
   (parameter UpdateCounts)
)

(class ReportReflectionsReceived reliable receive
  (parameter TransportationType)
   (parameter ReflectCounts)
)

(class ReportInteractionsSent reliable receive
  (parameter TransportationType)
   (parameter InteractionCounts)
)

(class ReportInteractionsReceived reliable receive
  (parameter TransportationType)
   (parameter InteractionCounts)
)

(class ReportObjectInformation reliable receive
  (parameter ObjectInstance)
   (parameter OwnedAttributeList)
   (parameter RegisteredClass)
   (parameter KnownClass)
)

(class Alert reliable receive
  (parameter AlertSeverity)
   (parameter AlertDescription)
   (parameter AlertID)
)

(class ReportServiceInvocation reliable receive
  (parameter Service)
   (parameter Initiator)
   (parameter SuccessIndicator)
   (parameter SuppliedArgument1)
   (parameter SuppliedArgument2)
   (parameter SuppliedArgument3)
   (parameter SuppliedArgument4)
   (parameter SuppliedArgument5)
   (parameter ReturnedArgument)
   (parameter ExceptionDescription)
   (parameter ExceptionID)
```

```
 )

)

 (class Adjust reliable receive
 (class SetTiming reliable receive
   (parameter ReportPeriod)
 )

 (class ModifyAttributeState reliable receive
   (parameter ObjectInstance)
    (parameter Attribute)
    (parameter AttributeState)
 )

 (class SetServiceReporting reliable receive
   (parameter ReportingState)
 )

 (class SetExceptionLogging reliable receive
   (parameter LoggingState)
 )
)

 (class Service reliable receive
 (class ResignFederationExecution reliable receive
   (parameter ResignAction)
 )

 (class SynchronizationPointAchieved reliable receive
   (parameter Label)
 )

 (class FederateSaveBegun reliable receive
)

 (class FederateSaveComplete reliable receive
   (parameter SuccessIndicator)
 )

 (class FederateRestoreComplete reliable receive
   (parameter SuccessIndicator)
 )
```

```
(class PublishObjectClass reliable receive
   (parameter ObjectClass)
    (parameter AttributeList)
)

(class UnpublishObjectClass reliable receive
   (parameter ObjectClass)
)

(class PublishInteractionClass reliable receive
   (parameter InteractionClass)
)

(class UnpublishInteractionClass reliable receive
   (parameter InteractionClass)
)

(class SubscribeObjectClassAttributes reliable receive
   (parameter ObjectClass)
    (parameter AttributeList)
    (parameter Active)
)

(class UnsubscribeObjectClass reliable receive
   (parameter ObjectClass)
)

(class SubscribeInteractionClass reliable receive
   (parameter InteractionClass)
    (parameter Active)
)

(class UnsubscribeInteractionClass reliable receive
   (parameter InteractionClass)
)

(class DeleteObjectInstance reliable receive
   (parameter ObjectInstance)
    (parameter Tag)
    (parameter FederationTime)
)

(class LocalDeleteObjectInstance reliable receive
   (parameter ObjectInstance)
```

```
)

(class ChangeAttributeTransportationType reliable receive
   (parameter ObjectInstance)
    (parameter AttributeList)
    (parameter TransportationType)
)

(class ChangeAttributeOrderType reliable receive
   (parameter ObjectInstance)
    (parameter AttributeList)
    (parameter OrderingType)
)

(class ChangeInteractionTransportationType reliable receive
   (parameter InteractionClass)
    (parameter TransportationType)
)

(class ChangeInteractionOrderType reliable receive
   (parameter InteractionClass)
    (parameter OrderingType)
)

(class UnconditionalAttributeOwnershipDivestiture reliable receive
   (parameter ObjectInstance)
    (parameter AttributeList)
)

(class EnableTimeRegulation reliable receive
   (parameter FederationTime)
    (parameter Lookahead)
)

(class DisableTimeRegulation reliable receive
)

(class EnableTimeConstrained reliable receive
)

(class DisableTimeConstrained reliable receive
)

(class EnableAsynchronousDelivery reliable receive
```

```
 )

 (class DisableAsynchronousDelivery reliable receive
)

 (class ModifyLookahead reliable receive
   (parameter Lookahead)
 )

 (class TimeAdvanceRequest reliable receive
   (parameter FederationTime)
 )

 (class TimeAdvanceRequestAvailable reliable receive
   (parameter FederationTime)
 )

 (class NextEventRequest reliable receive
   (parameter FederationTime)
 )

 (class NextEventRequestAvailable reliable receive
   (parameter FederationTime)
 )

 (class FlushQueueRequest reliable receive
   (parameter FederationTime)
 )
)
)
)
  )
 )
)
```

APPENDIX 7

```
;; BEGIN_INTERNAL_USE_ONLY -*- Lisp -*-
;; END_INTERNAL_USE_ONLY

;; RTI Next Generation RID (Run-Time Initialization Data) File
;; ============================================================
;; This file contains configuration parameters that control the operation of the
;; RTI software.  All parameters have a default setting that is used in the
;; event that a parameter value is not specified in the RID file or a RID file
;; is not specified.
;;
;; File Location
;; =============
;; The RTI-NG software looks for the environment variable, RTI_RID_FILE, which
;; defines the name and location of the RID file to be used by the application.
;; The file location may be absolute or relative using the appropriate file
;; naming convention for the particular operating system.  The file name is not
;; required to have a special name or prefix, it only needs to be readable by
;; the application and provide the correct syntax.
;;
;; If the environment variable is not set, the RTI will attempt to open a file
;; named "RTI.rid" in the directory from which the application was launched.
;;
;; File Format
;; ===========
;; The format used for the RID file has only several rules relative to valid
;; parsing.  The first item is that any text to the right of the comment token,
;; (two semi-colons, ";;"), is ignored by the parser.  The next rule is that the
;; left and right parentheses are used for scoping and must always be used in
;; matching pairs.
;;
;; Within a pair of parentheses there can be the scope name or a parameter name
;; and value pair.  The scope name is used to organize parameters that are
;; conceptually related and ensure uniqueness in case a parameter name is used
;; multiple times within different scopes.  If a parameter name is not unique
;; only the last value will be used for the configuration control.  The
;; parameter name is case insensitive and the value is parsed as a character
;; string and subsequently interpreted according to the particular parameter
;; type (e.g., integer, floating point, string).
;;
;; Parameter Scoping
```

```
;; ================
;; Each RID parameter is identified by a scope name in which the scoping is
;; broken into three major categories according to the granularity of the
;; internal RTI components.  The RTI-NG instantiates components when an RTI
;; process is initially started (the first create or join), when a federation
;; comes into existence within the process (first create or join of a new
;; federation), and when a particular federate joins a federation.  These scope
;; names are defined below.
;;
;;  ProcessSection    - process level component parameters
;;  FederationSection - federation level component parameters
;;  FederateSection   - federate level component parameters
;;
;; It is possible that a RID file used by a particular application will need to
;; support multiple federations and federates within a single process using
;; different RID parameter values for each federation or federate.  This RID
;; structure can support this situation by creating a scope within the
;; federation or federate section with the scope name the same as the name of
;; the federation or name of the federate, respectively.
;;
;; As an example, assume that an application needs to support two different
;; federations named FederationA and FederationB.  The RID parameter for the
;; multicast base address for FederationA needs to be different from
;; FederationB.  An example RID is shown below where the BaseAddress used for
;; FederationB is "224.100.0.1" and for any other federations the value is
;; "224.2.0.1".
;;
;;  (FederationSection
;;     ...
;;     (BaseAddress 224.2.0.1)
;;     ...
;;     (FederationB
;;        ...
;;        (BaseAddress 224.100.0.1)
;;        ...
;;     )
;;  )
;;
;; Parameter Definition
;; ====================
;; Each parameter contained in the RID file provides a description of the effect
;; that the parameter value has on the operation of the RTI.  The valid
;; parameter values are defined and the default value is specified within this
;; file.  As previously mentioned, if the parameter and value is not specified
```

```
;; within the RID file the default value will be used by the RTI.

(RTI
 ;; The RTI scope serves as a namespace for the RID user parameters.  No
 ;; parameter entries should be made at this level.

 (ProcessSection
  ;; Entries in this section apply to the process level components.

  (RtiExecutive
   ;; The RTI Executive is a logically centralized process that is used as a
   ;; network wide resource manager to handle such items as the uniqueness of
   ;; federation names.  It is logically centralized since redundant processes
   ;; can be used for fault tolerance (although this feature is currently not
   ;; supported).  The parameters associated with the RTI Executive control
   ;; how the process is found on the network.

   ;; PARAMETER: ProcessSection.RtiExecutive.RtiExecutiveEndpoint
   ;; DESCRIPTION: The RTI Executive endpoint defines the network address and
   ;; port number used by the RTI Executive process (and hence the RTI Naming
   ;; Service).  The network address can be a hostname or an IP address.  The
   ;; endpoint is only necessary when the multicast discovery mechanism is not
   ;; used and the endpoint must match the value provided when the RTI Executive
   ;; process is started.
   ;; RANGE: A valid hostname or IP address followed by a colon and then the
   ;; port number.
   ;; DEFAULT VALUE: None, will use multicast discovery mechanism.
   ;;
   ;; NOTE FROM HLA CERTIFICATION AGENT:
   ;; Change hostname to your hostname (do not use an IP Address).  Change the
   ;; port_number to either 5996 or 18134 (or some other high TCP port you would
   ;; like to use).
   ;;
   ;; Added for HLA Compliance Testing
   (RtiExecutiveEndpoint shannon-grad:5996)
   ;; End Added for HLA Compliance Testing

   ;; PARAMETER: ProcessSection.RtiExecutive.
   ;;          RtiExecutiveMulticastDiscoveryEndpoint
   ;; DESCRIPTION: The RTI Executive discovery parameter defines the multicast
   ;; address and port number used for the multicast discovery protocol to find
   ;; the RTI Naming Service which is located in the RTI Executive process
   ;; The naming service will then enable the application to locate distributed
   ;; RTI components (e.g., RTI Executive).
```

```
  ;; RANGE: A valid multicast IP address (or hostname) followed by a colon and
  ;; then the port number.
  ;; DEFAULT VALUE: 224.9.9.2:22605
  ;;
  ;;;;   (RtiExecutiveMulticastDiscoveryEndpoint 224.9.9.2:22605)

  ;; PARAMETER:
ProcessSection.RtiExecutive.NumberOfAttemptsToFindRtiExecutive
  ;; DESCRIPTION: The NumberAttemptsToFindRtiExecutive parameter is used to
  ;; control how many attempts the application should use to locate the RTI
  ;; Naming Service using the multicast discovery mechanism.
  ;; RANGE: An integer value greater than zero.
  ;; DEFAULT VALUE: 10
  ;;
  ;;;;   (NumberOfAttemptsToFindRtiExecutive 10)

  ;; PARAMETER:
ProcessSection.RtiExecutive.TimeToWaitAfterEachAttemptInSeconds
  ;; DESCRIPTION: The TimeToWaitAfterEachAttemptInSeconds parameter is used to
  ;; control how long the application should wait between attempts to find the
  ;; RTI Executive using the multicast discovery mechanism.
  ;; RANGE: A floating point value greater than zero.
  ;; DEFAULT VALUE: 2.0
  ;;
  ;;;;   (TimeToWaitAfterEachAttemptInSeconds 2.0)
  ) ;; End of ProcessSection.RtiExecutive

 (Networking
  ;; The Networking section is used to define the communication configuration
  ;; information associated with all of the RTI components within the
  ;; application using this RID file.

  ;; PARAMETER: ProcessSection.Networking.FederateEndpoint
  ;; DESCRIPTION: The Networking endpoint defines the network address and port
  ;; number used by the federate application process using this RID file.  The
  ;; network address can be a hostname or an IP address.  The federate endpoint
  ;; is used by other distributed RTI components to communicate with internal
  ;; modules within this application.  Typically the federate endpoint does not
  ;; need to be defined unless the computer has multiple network interfaces.
  ;; If an environmental variable named RTI_FEDERATE_ENDPOINT is found, its
  ;; value will be used in favor of what is specified here.
  ;; RANGE: A valid hostname or IP address followed by a colon and then the
  ;; port number.
  ;; DEFAULT VALUE: The default network card and the port.
```

```
;;
;;;;   (FederateEndpoint hostname:port)

(MulticastOptions
 ;; The networking multicast options define the parameters that control the
 ;; behavior of UDP communication within the RTI that is used for Best Effort
 ;; transport.

 ;; PARAMETER: ProcessSection.Networking.MulticastOptions.Interface
 ;; DESCRIPTION: The Interface is used to specify which ethernet
 ;; interface shall be used to send and receive multicast traffic. On
 ;; most systems the possible interfaces can be listed with the netstat
 ;; command).  If no interface is specified, the default is used.
 ;; NOTE: This parameter does not effect multicast name service discovery.[stb1]

 ;; DEFAULT VALUE: None.
 ;;
 ;;;;   (Interface "eth0")

 (Fragmentation
  ;; The UDP communication protocol (used for Best Effort transport) does
  ;; not fragment and reassemble data.  For messages larger than the UDP
  ;; fragmentation size the RTI must fragment the message into smaller
  ;; packets on the send side and then reassemble the packets on the
  ;; receiver side.
```

APPENDIX 8

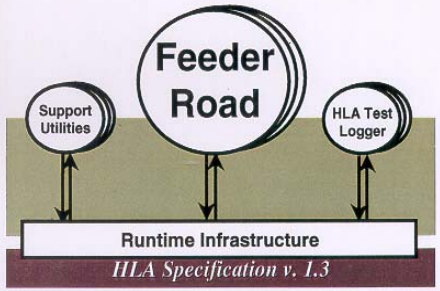
High Level Architecture Certificate of Compliance

This is to certify that

Feeder Road (CORSIM) (Federate Name) **Version 1.0** (Federate Version)

Successfully completed HLA conformance on August 19, 2002,

and is certified to be HLA-compliant in accordance with the HLA Specification v. 1.3.



Michael G. Lilienthal

Michael G. Lilienthal
CAPT MSC USN
Director, Defense Modeling and Simulation Office

VITA

Sharif Melouk received a Bachelor of Science degree in mechanical engineering from Oklahoma State University in 1993. He also received a minor in mathematics. In August 1995, he joined the College of Business graduate program at Oklahoma State University. He was granted a Master of Business Administration degree in May 1997. He joined the industrial engineering doctoral program at Texas A&M University in September 1997, where he studied and carried out research under the guidance of Professor Robert E. Shannon.

Sharif Melouk can be reached at the following address:

Department of Industrial Engineering

Texas A&M University

College Station, TX 77843-3131