THIN CLIENT DISTRIBUTED SIMULATION OF DISCRETE EVENT MODELS

By

Colin Timmons, BEng

A thesis submitted to

The Faculty of Graduate Studies and Research

In partial fulfillment of the
requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario
Canada

The undersigned hereby recommends to the Faculty of Graduate Studies and Research acceptance of the thesis

Thin Client Distributed Simulation
of Discrete Event Models

Submitted by Colin Timmons

In partial fulfillment of the requirements of the
Degree of Master of Applied Science

_____

Thesis Supervisor
Dr. Gabriel Wainer

_____

Acting Chair, Department of Systems and Computer Engineering
Dr. Jérôme Talim

Carleton University
2013

ABSTRACT

Distributed simulation is defined as real-time functional or platform-level war-gaming on multiple host computers. This simulation is composed of self-sufficient monolithic applications designed to communicate only across a standard interface such as High Level Architecture (HLA), Testing and Training Enabling Architecture (TENA), the Distributed Interactive Simulation (DIS) protocol or Link 16. The applications may represent up to Command, Control, Communication, Computer, Intelligence, Surveillance, and Reconnaissance (C4ISR) simulation applications. The Discrete-EVent System Specifications (DEVS) formalism has been extensively to study discrete event systems. Cellular-Discrete EVent System Specification (Cell-DEVS) is a DEVS-based formalism that combines cellular automata and DEVS allowing implementation of cellular models with timing delays. CD++ is a modeling and simulation toolkit used to build DEVS and Cell-DEVS models with a RESTful web-based interface in a distributed systems context. The distributed simulations and distributed systems are incompatible to directly interface which each other without refactoring either side. A thin client technique has been proposed to allow interoperability between the two M&S frameworks.

This work studies HyperText Markup Language – version 5 (HTML5), JavaScript, multi-threading, WebSockets, the presentation and extraction of Cell-DEVS data, the interfaces of distributed simulations and the WebLVC protocol in order to expose the Cell-DEVS data housed on the RESTful Interoperable Simulation Environment (RISE) into a distributed simulation. As an example, it presents the design and implementation of a web-based thin client as a mash-up of services between the RISE service and the distributed simulation. The proposed web-based client technique takes advantage of web service technologies in order to execute complex Cell-DEVS models within a distributed environment. In addition, the technique exposes distributed simulations to other simulation types beyond the Cell-DEVS discrete event models. The web-based client technique depends on the network connectivity but relies on the HTML5 WebSocket for providing full-duplex communications over a single Transport Control Protocol (TCP) connection to the distributed simulation.

Acknowledgments

This work is dedicated to my family for their endless support
and care. I also would like to thank Dr. Gabriel Wainer for his advice and guidance.
Specifically, the Aerospace Engineering Council for the recommendation and
the Canadian Forces Warfare Center for the freedom
and opportunity to pursue to this endeavour.

Table of Contents

List Of Figures

List of Tables

Acronyms

2D	Two Dimensional
3D	Three Dimensional
AJAX	Asynchronous JavaScript and eXtensible Markup Language
ALSP	Aggregate Level Simulation Protocol
API	Application Programming Interface
ARCGIS	Aeronautical Reconnaissance Coverage Geographic Information System
ASCII	American Standard Code for Information Interchange
ASTT	Advance Simulation Technology Thrust
BLOB	Binary Large Object
BOM	Base Object Model
C4ISR	Command, Control, Communication, Computer, Intelligence, Surveillance, and Reconnaissance
CD++	Toolkit for discrete event modeling and simulation
CDSC	Central Distributed Simulation Component
Cell-DEVS	Cellular Discrete Event System Specification
CORS	Cross-Origin Resource Sharing
COTS	Commercial-Off-The-Shelf
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DARPA	Defense Advanced Research Projects Agency
DCD++	Distributed CD++
DEVS	Discrete Event System Specification
DIS	Distributed Interactive Simulation
DoD	United States Department of Defense
DOM	Document Object Model
DNS	Domain Name System
DRY	Don't Repeat Yourself
DSF	Distributed System Federate

DSS	Distributed Simulation Server
DTD	Document Type Definition
EE	Enterprise Edition
ESRI	Environmental Systems Research Institute
FA	File Application Programming Interface
FOM	Federation Object Model
GB	Gigabyte
GMT	Greenwich Mean Time
GPU	Graphical Processing Unit
HLA	High Level Architecture
HTML	HyperText Markup Language
HTML5	Fifth revision of HTML
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDE	Integrated Development Environment
IE	Internet Explorer
IEEE	Institute of Electrical and Electronic Engineers
I/F Spec	Interface Specification
IP	Internet Protocol
ISO	International Organization for Standardization
JNI	Java Native Interface
JSON	JavaScript Object Notation
kB	kilobytes
LVC	Live, Virtual and Constructive
M&S	Modeling and Simulation
MA	Text file containing the Cell-DEVS specification in CD++
MEASURE	Mission Effectiveness Analysis Simulator for Utility, Research and Evaluation
NaN	Not A Number
REST	Representational State Transfer
RISE	RESTful-CD++ Interface Simulation Environment
P-DEVS	Parallel DEVS

PDA	Personal Digital Assistant
PDU	Protocol Data Unit
RFC	Request for Comments
RPC	Remote Procedural Call
RPR-FOM	Real-time Platform-level Reference Federation Object Model
RTI	Run-Time Infrastructure
SGML	Standard Generalized Mark-up Language
SIMNET	SIMulation NETworking
SISO	Simulation Interoperability Standards Organization
SOAP	Simple Object Access Protocol
SOM	Simulation Object Model
SRP	Single Responsibility Principle
TENA	Test and Training Enabling Architecture
TCP	Transmission Control Protocol
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
UTF-8	Universal character set Transformation Format – 8 bit
VR	Virtual Reality
VSIM	Virtual Simulation
W3C	World Wide Web Consortium
WebGL	Web Graphics Library
WS	Web Service
WSB	Web Service Bridge
WSS	Web Service Server
WTP	Web Tools Platform
WW	Web Workers
WWW	World Wide Web
XHTML	eXtensible HyperText Markup Language

XML	eXtensible Markup Language

Chapter 1: Introduction

Engineering and science communities have become increasingly aware that computer simulation is an indispensable tool for resolving multitudes of scientific and technological problems. Overwhelming concurrence is that simulation is fundamental to achieving progress in engineering and science in the foreseeable future. Engineering and science disciples have agreed that the computational and simulation engineering science are also fundamental to the security and welfare of a nation [SBES06].

As simulations model real-world conditions, simulation technology must continue to evolve as world events are often unpredictable. In defence terms, the defence industry must be able to design, modify and train in quick response to both military and police demands of force. While beneficial, it is not always fiscally responsible, time permitted or safe to experiment and train personnel in a physical joint or combined force concept.

High Level Architecture (HLA) is the standard for distributed simulations for providing the interoperability between and among defence forces and is becoming prominent within the civilian simulation systems [Str01]. It has been used to study and develop tactics through the joining and operation of multiple simulators. These defence simulations have always been held behind secure network infrastructures and remain dependent on the requirements of the military environments and its development of specific software and network connectivity. However, not all defence simulations have been developed with the capability to interoperate in distributed simulations or are not supported by maintenance and remain fixed at a certain level of robustness. These systems remain isolated and incapable of participating within a traditional distributed simulation such as a Distributed Interactive Simulation (DIS) protocol environment or HLA federation. In order to provide training and tactics to these valuable crews, one is forced either to physically train the personnel in the non distributed simulation trainer or to physically train the personnel

using serviceable equipment such as aircraft, ships or tanks in consort with other joint or international combined forces.

With the advances of technology, simulations now can be accessed via Personal Digital Assistant (PDA) such as cell phones, tablets, and similar devices. No longer are simulations constrained to desktop and embedded user interfaces. The World Wide Web (WWW) has become a major platform for complex and sophisticated enterprise applications and services across domains. Simple Object Access Protocol (SOAP) based web servers have traditionally relied on the server to provide the fidelity and utility for the user and allowed the client side to remain as an interface to the server applet. Advances in hardware, wireless and networking and software technology have provided a means to provide an interface for isolated simulations and to further provide high-fidelity resources to traditional distributed simulations.

Discrete-EVent System Specifications (DEVS) is a Modeling and Simulation (M&S) formalism that has been used to study discrete systems [Zei00]. By its nature, DEVS provides a sound M&S framework, supports a full range of dynamics system representation capabilities, and supports a hierarchical, modular model development. The external outputs, internal inputs, and model states are dependent as a response to external events and are mathematically defined for the model's behaviour.

Follow-on to the DEVS and relying on its formalism is Cellular-Discrete EVent System Specification (Cell-DEVS) [Wai04]. Cell-DEVS is an extension that allows n-dimensional representation of each cell as a DEVS model that only is activated by external stimuli from neighbouring cells within its cell space. By individualizing the cell space, zones of different behaviour and variable timing parameters or delays can be created. Thus, Cell-DEVS provides the means to understand and control scalable, multi-physics phenomena in n-dimensional directions through its M&S framework. Examples of successful complex systems that have been modeled are the fire spread in a forest [Ame01], a battlefield conflict [Mad05] and computer networks [Ahm05].

REpresentational State Transfer (REST) [Fie00] is a style of software architecture for distributed systems such as the WWW. REST is a set of principles or constraints that define how Web standards are supposed to be utilized. The principles are that every resource has a unified concept as a globally unique Identifier, that the Uniform Resource Identifier (URI) resources are linked together and grant the client the ability to manage states rather than the server, that every resource supports the same set of methods as defined in the HyperText Transfer Protocol (HTTP) specification, that there is separation of handling data and invoking operations through multiple representations and statelessness as it isolates the client against changes on the server. By this, statelessness infers scalability of services. A server that follows the constraints of REST is termed RESTful.

RESTful Interoperable Simulation Environment (RISE) is a middleware implementation of a RESTful-CD++ [Wan12]. CD++ [Wai02] is an M&S toolkit that was developed to execute DEVS and Cell-DEVS models on stand-alone machine. Distributed CD++ (DCD++) [Hou06] implemented a distributed CD++ framework that allowed Web Services (WS) technologies to employ multiple heterogeneous computer resources. Originally, by establishing network connectivity among the machine, different simulators could exchange messages during the distributed system session using SOAP.

RISE creates a Restful-CD++ service to the user, proving that distributed simulation synchronization is achievable via purely using eXtensible Markup Language (XML) messages, which enables the developers to hide internal implementation. By supplying DCD++ service resources, DCD++ is enabled to be part of the mash-up of the Web, advancing interoperability to a higher level than a SOAP-based simulation WS. This is because the Restful-CD++ handles incoming requests to its resource Application Programming Interface (API) as URIs, and hence become independent of service type.

The modeller client, from the viewpoint of a webpage, only connects to the resource API of the main server through its URIs. The main server instructs the supportive servers to hide all its resources from external users. By separating the middleware from services, RISE provides the benefit of granting the ability to migrate the DCD++ simulation services to

new hardware topology, the modularity in support simulation services, the control of baseline versioning of the simulation engines, and the simulation URIs serve as simulation engine wrappers.

Typically, a thin client such as a browser has been used as a terminal to access a server. They have been designed to improve security and reduce hardware as the content and control is hosted by the server. Usually, the devices and software are simple but limited but as more and more capability is added, the expense increases. Capability such as three dimensional (3D) imaging, video conferencing and multi-monitor support may affect the display processing that the back end server can supply. In this configuration, a thin client is stateless, connected to a server who controls the resources and computing experience.

However, new technologies and standards have been developed in the WWW which enable highly-interactive, low-latency, real-time web based applications written in JavaScript [MÄk12]. These technologies include the HyperText Markup Language – version 5 (HTML5) canvas element, HTML5 video element, Web Graphics Library (WebGL), WebSocket, and new JavaScript libraries. The HTML5 canvas element allows dynamic, scriptable rendering of two Dimensional (2D) shapes and bitmap images, while HTML5 video element allows a standard for embedding a video or movie rather than a third party plug-in. WebGL is a JavaScript API for rendering interactive 3D and 3D graphics within any compatible web browser without the use of a plug-in. WebSockets are a JavaScript interface which defines a full-duplex single socket connection between the client and the server. The new HTML5 API allows for interactive dynamic client application with the ability for file service interaction.

HTML5 promotes the importance of rich thin client rather than the proprietary thin or zero clients. It enforces the understanding that the web is accessed by billions of people and has become vital for commerce, education, entertainment, and information sharing. It has become a central and common item is people's daily life allowing interoperability and linkages to sites, data, and knowledge. As the world turns to the web more and more, with

the power of the cloud, the security of virtual machines, companies like Amazon, Facebook, and the Financial Times, to name a few, are pioneering the power and simplicity of HTML5. Previously, vendors have been releasing patches to address vulnerabilities in their installed applications to limit and minimize the risk of security risks from third-party plug-ins. With HTML5 and the ability to create zero-footprint applications, less network activity is possible and subsequently faster uploads/downloads with quicker display updates.

However, because of their very nature and construction, distributed simulations and distributed systems are not interoperable unless an agreed interoperability standard is used such as HLA. Neither component have an open API accessible by each other and distributed simulations are designed with rigorous and specialized components that enable synthetic training environments. There is currently no standard interoperability protocol to adequately link new web-based applications with each other and with traditional distributed simulations. In order to connect and allow interoperation, and from the perspective of a federation wanting any non-distributed simulation to join, modifications would have to be composed into the federation by the addition of a Web Services Server (WSS) for HLA 1516-2010, with the addition of a Web Services Bridge (WSB) for HLA 1516-2000 and older. This is demonstrated in Figure 1 while Figure 2 demonstrates the interoperability problem of a distributed system federate interfacing with older versions of HLA such as HLA 1516-2000. In both cases an interface must be designed that translates the web service information into the distributed simulation. It must be noted that the information to be transferred between the systems has yet to be defined; only the availability of the data is present and in this case, only for HLA.

Similarly, by leaving the distributed simulation alone, another technique is to compose the non-distributed simulation into a federate as shown in the right of Figure 3. In this technique the non-distributed simulation is refactored to publish data of interest to a stub, called the Distributed Simulation Server (DSS) through a socket and the DSS transmits and receives the data through its interface with the distributed simulation. Again in this case, the information to be transferred has yet to be defined. In this case, the Distributed System Federate (DSF) data of interest would have to be exposed and the non-distributed

simulation refactored to export the data for publication to the DSS. Again the content of the data has not been detailed just that the data could be available.

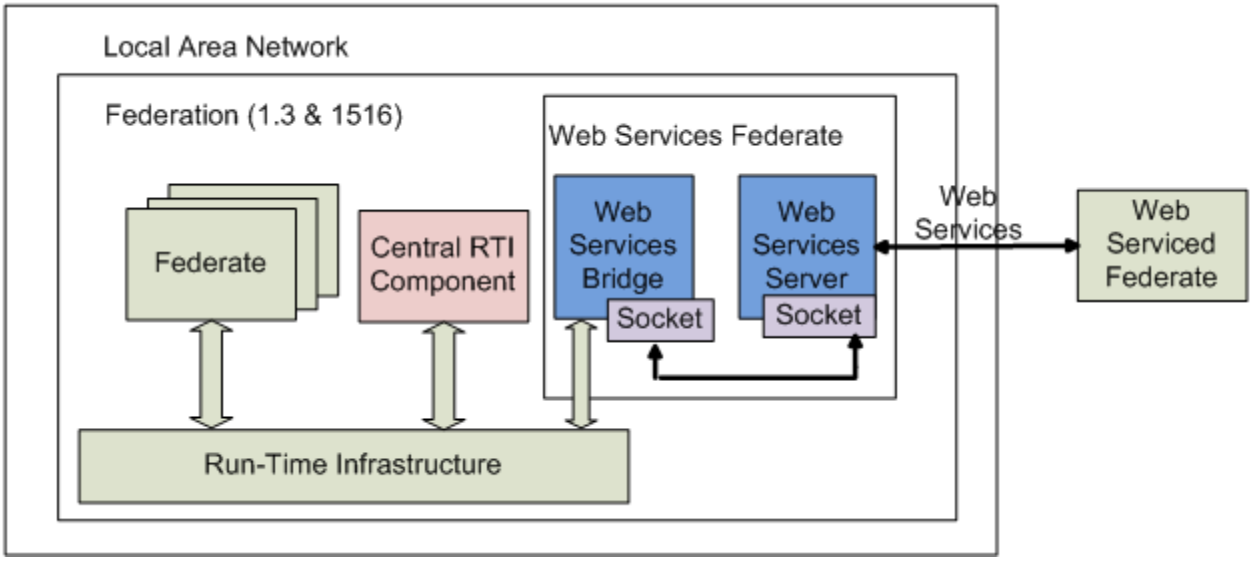

Figure 1. Web-Enabled Simulation Joining a HLA 1516-2010 Distributed Simulation.

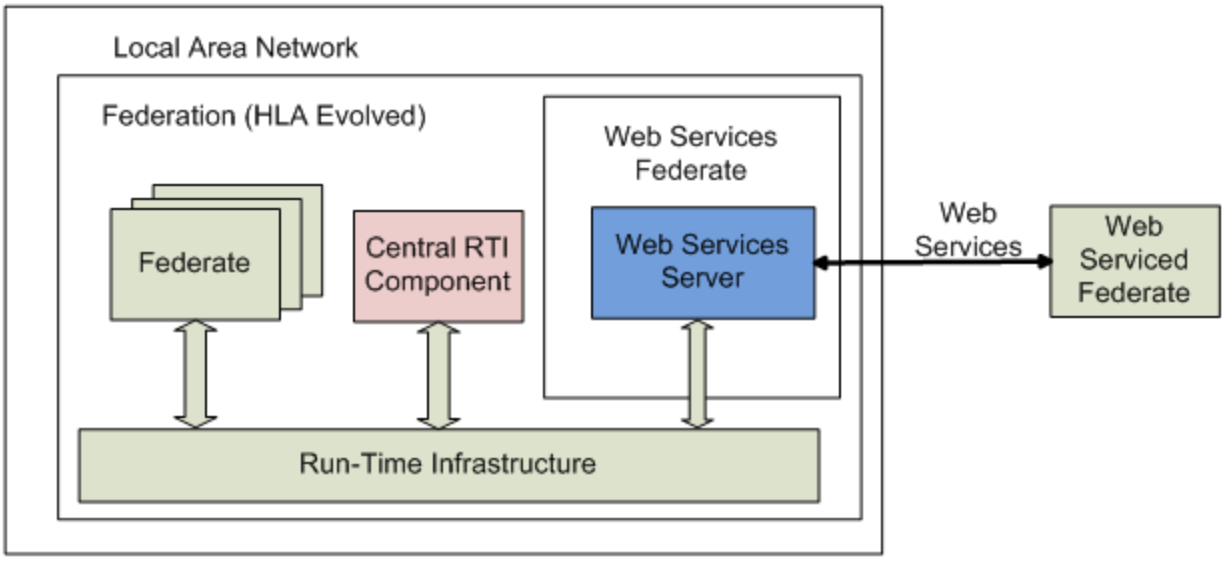

Figure 2. Web-Enabled Simulation Joining a HLA 1516-2000 Distributed Simulation.

However, by leveraging the new technologies of the WWW and its cultural demands [Pag98], a technique can be created that perform the duties of interoperability. Central to the above figures is the commonality of a Central Run-Time Infrastructure (RTI) Component and WS/DSS which can be isolated and refactored into a Web-Enabled Server

(WES) as shown in the right of Figure 4. With this in place, the non-distributed simulations are isolated by a separation of concerns from the Web-Enabled Server and can either update or reflect data as produced by their original requirements. They can retain their ability to access other WS and populate as a mash-up of services composed of the other WS into the distributed simulation. However, the key to this modularity is the WES which houses the ability to translate WS data into the distributed simulation environment regardless of the protocol or the FOM specifics of the distributed simulation. It would be beneficial to have a Central Distributed Simulation Component (CDSC) that could handle and translate the different architectures and protocols of distributed simulation in order to remove the requirement of developing to the architectures and protocols specific to the architecture of the distributed simulation.

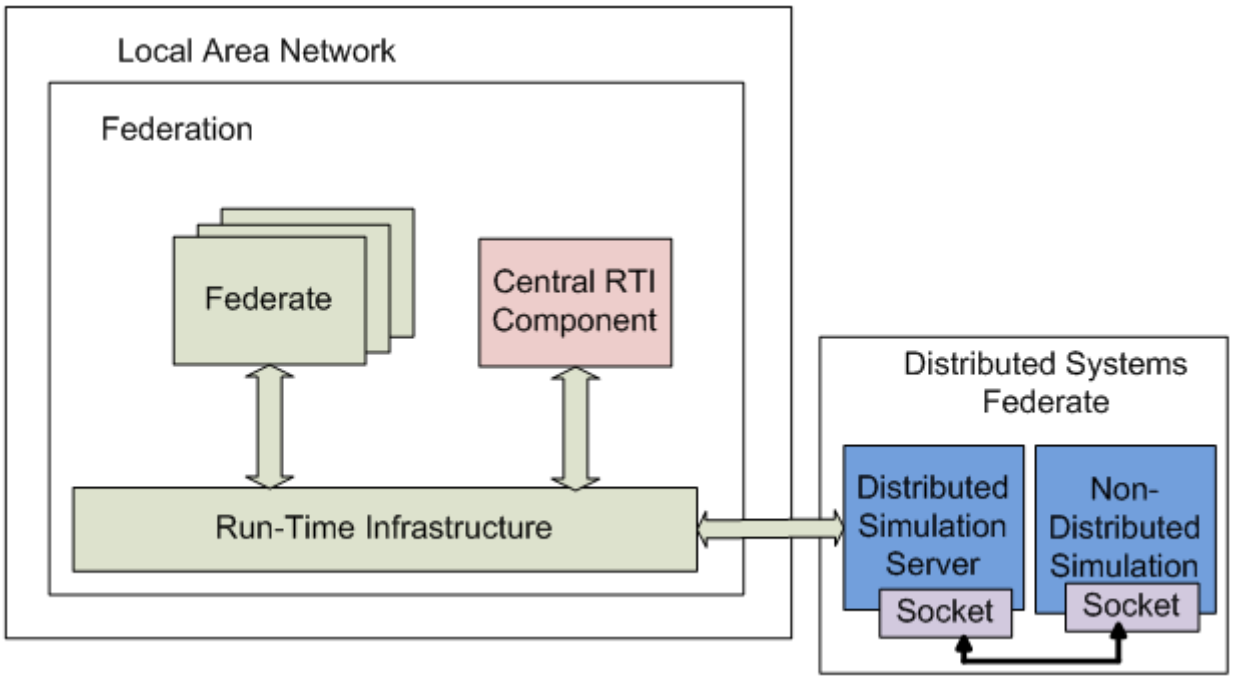

Figure 3. Non-Distributed Simulation Joining a Generic Distributed Simulation.

With respect to the WS capabilities and infrastructure, it would be beneficial to leverage new technologies like WebSockets for full-duplex communication rather than relying on Universal Datagram Protocol (UDP) and Transmission Control Protocol (TCP). In this case, a client simulation is able to create an asynchronous full-duplex channel to the host server. This communication allows the client to send data instantly to the server and have the

server communicate to the client at the same time while the connection is open.

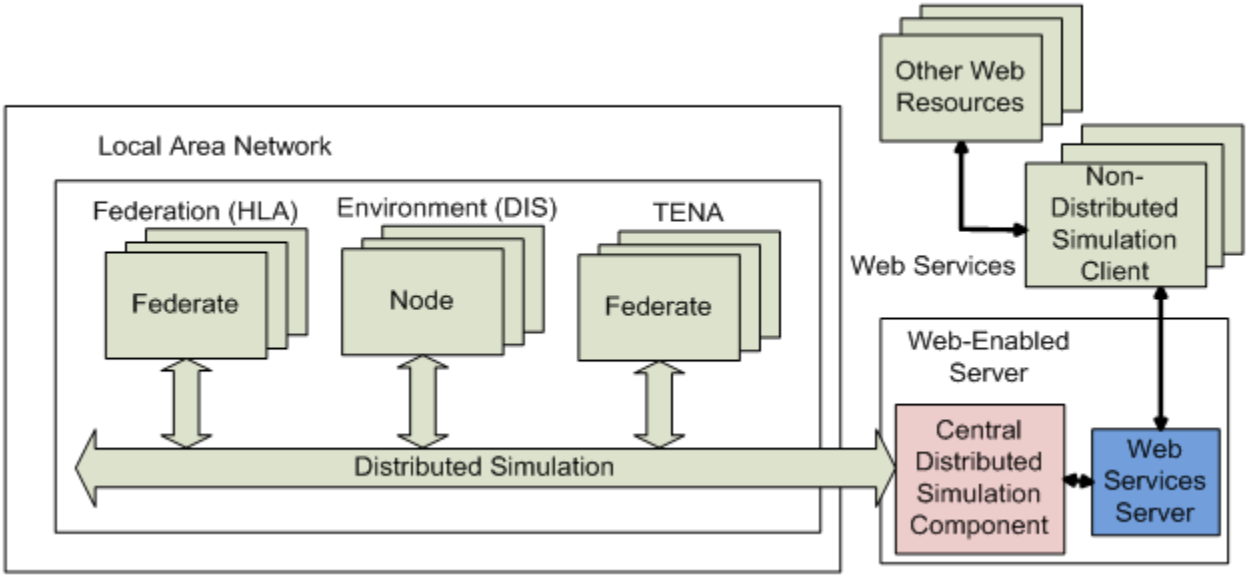

Figure 4. Thin Client Distributed Simulation Architecture.

MÄk took the initiative to create such a CDSC which would accept bi-directional data and proposed it as new standard to the Simulation Interoperability Standards Organization (SISO). This initiative is called the Web Live, Virtual and Constructive (WebLVC) protocol and harnesses these web technologies to support interoperability between thin clients and traditional distributed simulations which may be using DIS, HLA, Testing and Training Enabling Architecture (TENA) or any other distributed simulation architecture [MÄk12].

In order to provide testing and a platform for developing new interfaced simulations, MÄk has also created a WebLVC server to connect the web-based simulations into distributed simulations through the WebLVC protocol as shown in Figure 5 . As the WebLVC server is based on their commercial available Virtual Reality (VR)-Exchange interoperability portal broker [MÄk12], web based federates or nodes can now participate in distributed simulations by using the WebLVC protocol and the WebLVC server. Based on MÄk's VR-Link [MÄk12], JavaScript libraries have become available that offers the key functionality to operate and allow the implementation of the client side of the WebLVC protocol.

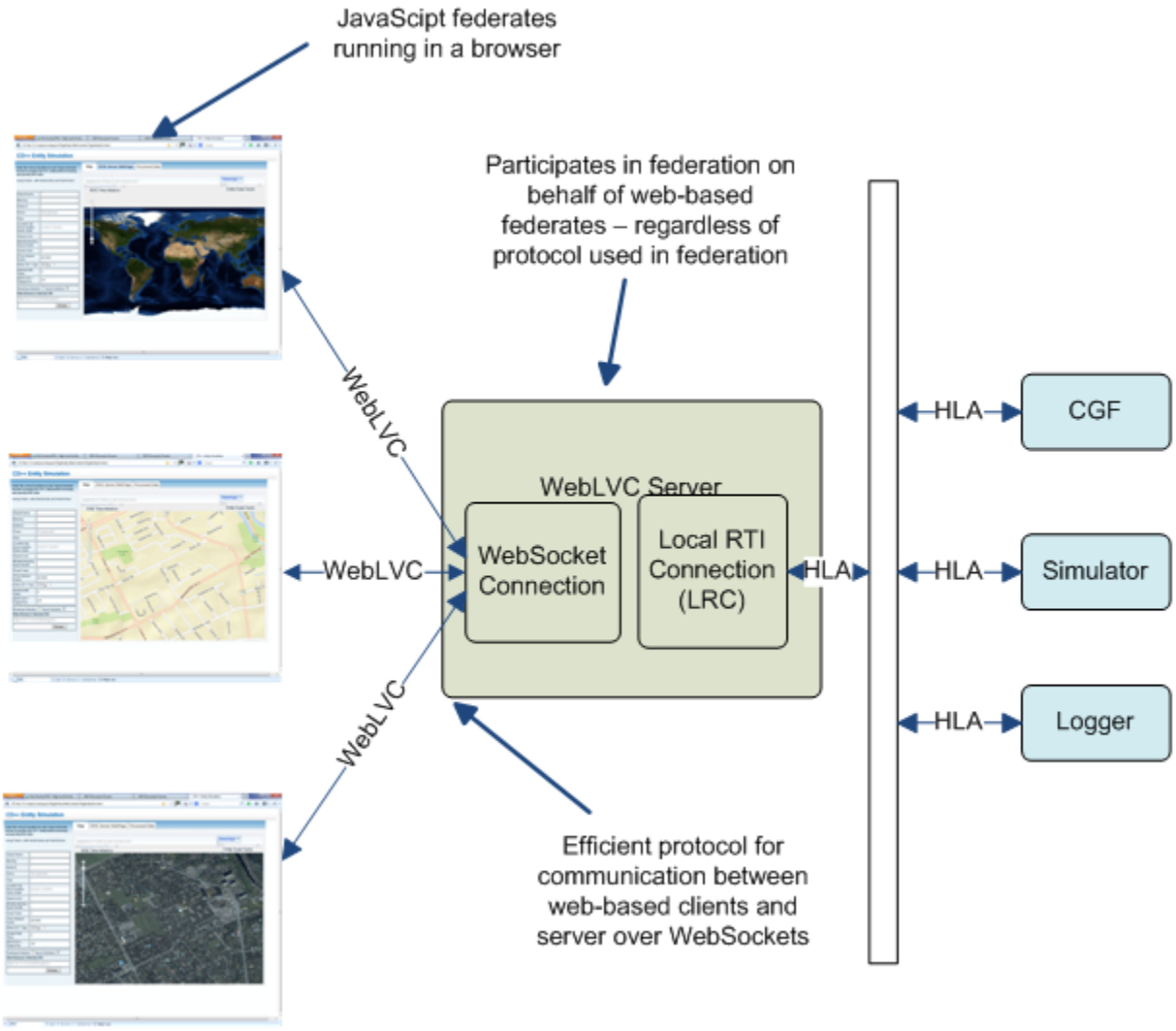

Figure 5. WebLVC Interaction Diagram.

Distributed simulations and in particular HLA do not offer the range of support for simulation model organization that is found within the traditional discrete event simulation conceptual framework. However, distributed simulation developers have realized the importance of discrete event simulation in order to develop functional-level federates. On the other hand, RISE is a plug and play interoperability paradigm of software tools and is accessible by a thin client through its resource API which is based on URIs. The thin client controls the state of RISE and thus becomes a requirement in order to leverage the traditional discrete event simulation conceptual framework execution results.

The key problem identified with RISE and distributed simulations is that they were designed based on separate requirements and therefore separate fields of operations. Distributed simulations cannot access the simulation execution results of RISE and similarly, RISE cannot populate its simulation execution results directly into distributed simulations. Discrete event models have been successfully interfaced together recently using HLA as the interoperable standard [Pri05]. However, the attributes, information and objects needed to transfer between one model and another was required prior to the development of the model's formalism and is based solely on the content being transferred. Thus, using the WebLVC Server as an access point to the distributed simulations, a thin client would act as a mash-up of services for the manipulation and generation of discrete event simulation data as distributed simulation entities.

The need to integrate the CD++ simulation capabilities into larger high fidelity user interfaces has become of topic of research [Wai10]. High-fidelity visualization tools such as 3DMax or Blender has aided the user in interpreting the simulation results and but enhanced the proficiency requirement of the user to understand the CD++ output. Similarly, dynamic terrain modeling of forest fires and floods dramatically enforces the critical thinking required for a battlefield commander to command and control his forces. The correct development of the commander's orders is dependent on his cognizance of the area and the expectation for realism. In using a web-based client to access the RESTFul-CD++ data, an added bonus is a means to inspect any simulation execution result in a non-textual manner as provided by the CD++ display tool.

The incentive of this work stems from the thrust to be able to introduce information into a distributed simulation without prior knowledge of the attributes, information or objects composing the non-distributed simulation. In this regard, the non-distributed simulations are web based. Similarly, the need to interface complex and dynamic modular computational systems into distributed simulations increases the fidelity of the distributed simulation. As natural and artificial systems evolve, the capability to integrate these systems with larger systems to provide a mash-up of services and resources will advance the use of simulation technology. Although many different interfacing techniques have

been attempted or are proprietary developed, they are specific to the hardware, software language, network connectivity requirements or content type. We aim at a novel manner in exposing a web based simulation for the use in a traditional distributed simulation environment through the thin client distributed simulation architecture. Thus, the new approach is to provide discrete event data from a complex simulation engine through a thin client over communication channels that allow full-duplex communications over a single TCP connection to the distributed simulation regardless of the specifics of the Federation Object Model (FOM) giving other discrete event modellers the capacity to design discrete event models specific for the distributed simulation.

The validity of this approach has been demonstrated openly as web applications have existed in thin client architecture for a few years. The web is full of applications such as games, photo editors and video players that are more rich and interactive than a website but less cumbersome and monolithic than a desktop application. As an added benefit, one of the main advantages of having a web based client is the ability to update and maintain the application without distributing and installing software on numerous client computers.

1.1 Motivation and Goals

1.1.1 Presentation of Foundational Structural Elements

As already mentioned, simulated systems are becoming increasingly sophisticated and increasing complex. In distributed simulations, focus has been based on the ontology components in order to achieve simulation reuse. The reuse, though, has occurred through the interoperation of uniformly structured proprietary M&S interface components. Its presentation of the data is specified as a foundational structural element, specifically an entity. This entity is defined as “any distinguishable person, place, thing, event or concept about which information is kept” [SISO07]. Because of the complexity of distributed simulations, research has been extensive on creating visual coding tools that allow the simulation to present its data in the correct data types and formatting [Ban10]. However, the execution stage of the tools is dependent on a proprietary RTI as in HLA or middleware such as TENA rather than the ability to generate pattern visualizations regardless of the content.

1.1.2 Manipulation and Extraction of RESTful-CD++ Data

Presently, the RESTful-CD++ data is housed on a simulation engine platform called RISE which is accessible by a user through HTML and XML messages. The user must manipulate the hyperlinks viewable on the HTML pages in order to access, control, execute and gather the results. In order to visually examine the results, the textual simulation data must be manually extracted and manipulated by the user into a form acceptable for third party software applications. The manipulation of the data is specific only to the model being examined and to the third party application's import type.

1.1.3 Force Generation through Discrete Event Simulation

RISE is not capable of presenting data in a format acceptable for traditional distributed simulations. It has neither a distributed simulation interface nor a middleware component that could adapt the data into a form acceptable for distributed simulation. Since the DCD++ simulation engine can model and simulate a complex system under study, focus should be applied to expose these DCD++ simulation results into distributed simulations. With the DCD++ simulation data present in a distributed simulation, training and force generation capability can be enhanced through a simulation simulating the actions and interactions of autonomous active entities or environments in a network.

1.1.4 Web Enabled Simulation in a Distributed Simulation

In summary, web enabled simulations are not designed to be interfaced in a distributed simulation and similarly, a distributed simulation is not designed to allow web enabled simulations. The purpose of this dissertation is to examine and study the distributed simulation requirements, web based simulations, and WS technologies in order to develop an understanding of how an interface component could be developed or adapted to endorse web based simulation data in a distributed simulation.

1.2 Contribution

In this dissertation, we present the following research contributions:

a. a discussion and examination of the different available architectures used in distributed simulations;

b. an examination of the problems of integrating web-based simulations into requirement-based monolithic applications of distributed simulations;
c. An examination of the coordination required to interface of web based discrete event simulation into distributed simulations;
d. A means to compose the discrete event simulation results such as the RESTful-CD++ simulation results into an interactive distributed simulation;
e. a proof of concept for the design and implementation of a thin client connected into a distributed simulation; and
f. an examination and corroboration of the WebLVC protocol.

1.2.1 Contribution Rationale

Distributed simulations are monolithic in nature and communicate by recognized interoperability standards. Web based clients are not naturally exposed into distributed simulations. By providing a technique to interface a discrete event simulation into a distributed simulation, we exemplify a means to enable a web based mash-up of services into the distributed simulation that was previously unavailable as shown in Figure 6.

Distributed simulation exercises are usually conducted presently as a DIS or HLA federated exercise where environmental conditions are often simplistic or denoted as syntactic markers in the exercise. Such examples of the markers are defined as craters, fires, wind and snow conditions, or ice. We compose a means for integrating the discrete event simulation results such as the RESTful-CD++ simulation results into an interactive distributed simulation depicted in Figure 6 through the use of a thin client to extract, manipulate, generate and display the simulation results. This technique can be used for real-time generation of objects such as flooding of a position or threat of a realistic advancing fire on a battalion's logistics supply depot. RESTful-CD++ allows the interaction to generate complex representations to provide a mash-up of services with visualizations of complex frameworks instead of the simple boundary markers.

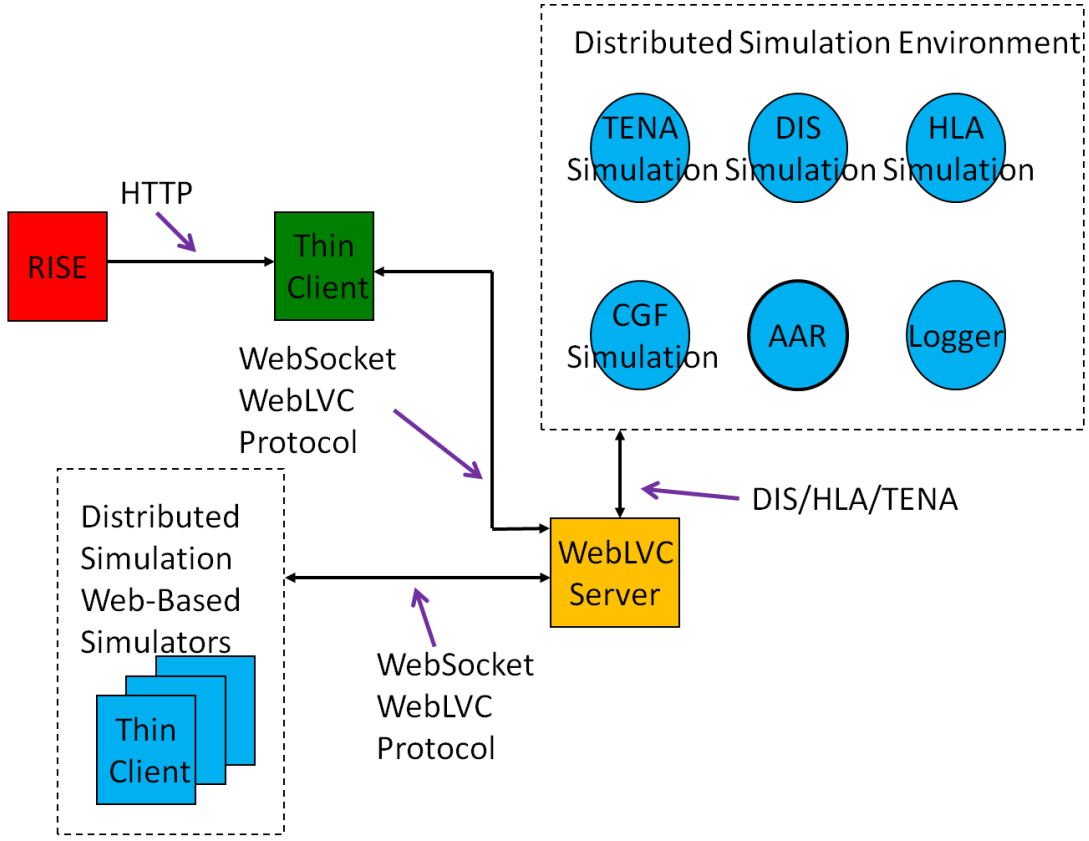

Figure 6. Thin Client Distributed Simulation of Discrete Event Models Architecture.

In particular, we present as a proof of concept the design and implementation of a thin client connected into a distributed simulation and using the model results processed by RESTful-CD++ and available via RISE. Firstly, an interactive client was designed that accesses the RESTful-CD++ simulation results and implements a user file access capability. A MÄk prototyped single entity simulation was re-designed to parse and extract Cell-DEVS data as multiple entities and to provide the capability for high-fidelity CD++ visualization. As Cross Origin Resource Sharing (CORS) has been a problem for client based applications and is still in the draft stage, and the mechanism to resolve client-side cross origin requests remain a modification to the server's response header, we have incorporated a mechanism for the user to interact with the resources and provide a user's requirement to activate the WebLVC interaction.

We also present an evaluation of the performance loading and usage of the thin client hosted on various machines. Because of the differences in browser versions and capabilities, the testing is dependent on the ability to support WebSockets, multi-threading and file access. Presently, Chrome is the only browser allowing permanent memory allocation, while Chrome, Firefox and Internet Explorer 10 support WebSockets, temporary memory allocation and multi-threading. The results of this technique were tabulated and are based on available commodity internet services.

1.4 Thesis Organization

This dissertation is organized into different chapters. In Chapter two, the different technologies that compose distributed simulations are detailed in order to provide the context for the difficulty in altering non-distributed simulations into distributed simulations In particular, we examine the RESTful architecture in general and directly, the RESTful-CD++ simulation environment and tools of RISE. Later, we will present the distributed simulations of HLA and DIS technologies and present the simulation of environmental entities. Next, we will provide an introduction to the new interfacing techniques proposed to enable non-distributed simulations to access distributed simulations. In particular, we examine the use of the MÄk WebLVC protocol and the MÄk WebLVC server. Lastly, we will describe the new web technologies required to create an interface technique to provide the interactive and dynamic services required for high fidelity and high resolution for a thin client.

Chapter three discusses the trend and evolution of both distributed simulations and the discrete event simulation engine platforms and the reason for the integration of the two as a mash-up of services. Chapter four introduces the technique to interface and for proof of concept, the design, implementation, architecture and interfaces of this technique. Chapter five discusses the development of the proof of concept to allow of a web based simulation to become a mash-up of services between a discrete event simulation and a distributed simulation. Significant in this chapter is the implementation layout of the design of the software and the functionality of the software. Also, we will present the experimental

results of the technique and its performance as a viable alternative to refactoring software for traditional distributed simulations. In Chapter six, we will present our conclusions and recommendations for future research that could enhance or extend the outcome of this dissertation. Chapter seven contains the references used for this dissertation.

Chapter 2: Simulation Environments and Tools

M&S environments are specific in terms of their implementation or use. In developing the requirements of the simulation environment, usually the users play an important part in developing the uses of the simulation. Some simulations are simplistic in nature and are based on strictly user response to an event queue or highly complex in computational manipulation. With regard to the fidelity of the simulations, a complex system may produce a single response output while an event queue simulation may produce dramatic real-life high fidelity from a user's perspective. The cost, resources, and scheduling of the simulation are based on the requirements and correctness of the simulation. A simple output of a complex computational program may require extensive debugging to discover the error in programming, or even modification to the mathematics behind the program. A high fidelity simulation will require continual user interaction to ensure the desired simulation is a reflection of the use cases and the communication involved to transfer the ideas of what is significant and how it should be implemented. The path to which a simulation evolves through its lifecycle is also dependent of the technologies present at the design time and is reflected in the implementation and testing of the simulation. Modern object oriented practices which define modularity and conciseness in design suggest that the either the output is robust as an open API or that code may be functionally separated into independent interchangeable modules such that only one module is necessary to execute one aspect of functionality. However, the intent of the modularization remains to provide a service dependent on the original requirements. In this chapter we will provide an overview of the technology that produced these services for the different M&S environments.

2.1 Development of RESTful-CD++

The RESTful-CD++ was developed over time based on the formalism of DEVS and the desire to provide accessibility to user for the M&S of complex natural and artificial systems in a flexible and simplistic manner. It is a natural evolution of the services developed to study discrete event systems and is a plug and play interoperability paradigm of software tools.

2.1.1 Discrete Event System Specification

Historically, the DEVS formalism has been extensively used to study discrete event systems. DEVS provides a sound M&S framework, supports a full range of dynamics system representation capability, and supports a hierarchical, modular model development [Wai10]. DEVS focuses on the changes of variable values as the system state and its transitions as events. It generates simulation time segments that are piecewise constant. Discrete-events models are defined by the occurrence of events where the event can be defined as an occurrence that happens instantly in a given point in time and affects the represented system. Each event computation contains a simulation time stamp indicating when the event occurs in the system. Thus, each event computation may either modify variables or schedule new events into the simulated future environment being simulated. However, the state of the model can only change a finite number of times in a finite time interval, depending on the occurrence of the events.

By definition, the mathematical language called formalism realizes how to generate new values for variables and when the new values should take effect. An extension to the DEVS formalism is Cell-DEVS which allows atomic and component construction. Atomic modularity construction provides the lowest level of model construction and contains the structural dynamics while component construction is composed of one or more atomic and or coupled models. Figure 7 [Wai11] represents the semantics and flow of the DEVS atomic model.

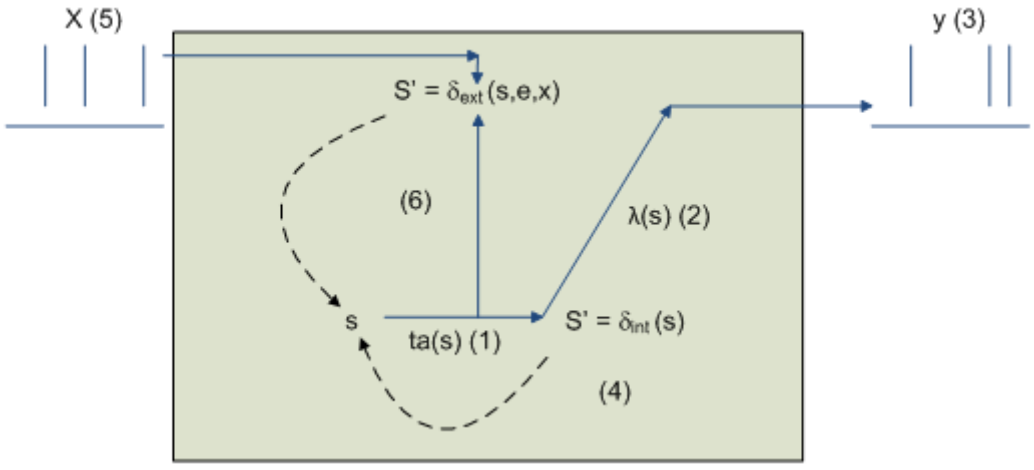

Figure 7. DEVS Atomic Model Semantics.

Briefly, an atomic model remains in the state s ϵ S, for a time t_s = ta(s). Before t_s has elapsed, an output can be constructed using the output function and the state previous to the event such that Y = λ(s). After t_s has elapsed, an internal event is triggered and the state is changed to $s_{new} = \delta_{int}(s)$. After the execution of the transition, a new internal event is scheduled at time $t_{s,new} = t_a,(s_{new})$ + some time. The DEVS formalism provides a number of distinct atomic behavioural characteristics such as input ports, output ports, state variables, state transition functions, output functions and time advance functionality. The coupled model characteristics provided are structural components, and the coupling of internal or external input and external outputs. One major importance of Cell-DEVS is that n-dimensional cell spaces can be created and utilized and n-dimensional zones of different behaviour can also be incorporated. Specifically, input and output events through specialized ports in the predefined cells allow robust and complex atomicity of interaction within its neighbourhood. Figure 8 [Mah05] illustrates the cellular automaton neighbourhood which is defined as a Moore Neighbourhood where the dots represent occupied grid cells.

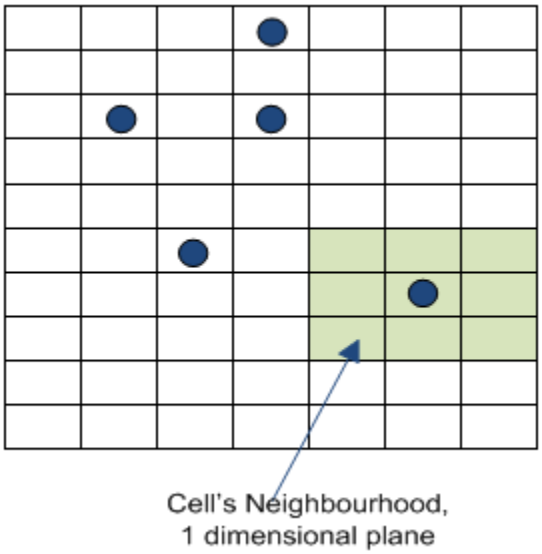

Figure 8. Cellular Automaton Neighbourhood

A cell can have different valid states in its lifecycle – empty, occupied, or as a static / moving obstacle. In order to define the model, the CD++ modeller needs to specify the cell space shape, size and the set of laws or rules governing the model execution which grant a

cell its distinct values. For example, Madhoun et al [Mah05] defined the Unmanned Aerial Vehicle (UAV) cell space state as empty (0), moving (1), a moving obstacle (5) and a static obstacle (9).

2.1.2 Modeling and Simulation Generation

The CD++ is an M&S toolkit following the definition of the DEVS which separates modeling from simulation and contains an extension to include Cell-DEVS models. Models that have been generated through CD++ modeling are processors, excitable media, surface tension, fluid dynamics, 3D heat diffusion, breast cancer, life, battle space, fire, and flood to name a few. With n-dimensional cell modeling, one would expect the complexity of the output to be hard to envision and it is. The CD++ toolkit generates a textual visual representation of the data on a 2D plane per cell-layer. Using 3D modeling strategies and specific to the model, one could manually interface the output files as input into another third party visualization tool such as 3DMax, Google Earth, or Blender.

CD++ was originally developed to execute sequentially on a single workstation. Serialization errors occur if the states of two different models connected together expire at the same time. Such an exception is left at discretion to the modeller to evaluate as to which model is to execute next. To overcome this issue, Parallel-DEVS (P-DEVS) was developed allowing all models with the earliest change state to execute in parallel granting the simulation the full benefit of parallelism. P-DEVS is a collection of sequential discrete event simulation engines executing on different computation platforms that communicate by sending time stamped event messages. The major difference between the DEVS and P-DEVS in formalism is the addition of a confluent function (δ_{conf}) which provides the ability to determine the next state of the model when an external input arrives at the same time as an internal transition.

2.1.3 Distributed CD++

DCD++ was created to perform distributed simulation between different CD++ engines. Its purpose was to allow partitioning of the simulation environment such that each engine is

responsible for a portion of the entire model and coordination between the models. Distributing the CD++ simulation engines allowed the resources of WS technologies to execute complex models and the parallel computation power of a distributed network.

As more and more simulations are being developed for complex models to represent natural and artificial systems, this need arose in order to provide a means to wrap the CD++ services as SOAP based WS. This service would provide remote access to the CD++ services allowing the user to remotely create a simulation environment, submit the model, start and execution the simulation and retrieve the results.

The main component that allows the WS aspect of DCD++ is the wrapper which is composed of a WS component, written in Java, and a simulation component, written in C++. The WS component services user authentication, session management and parsing simulation requests and is deployed as an Axis SOAP engine running via an Apache Tomcat application server [Mad06]. The Simulation component accesses and manipulates the internal objects and data structures in the simulation engine.

Client session interaction is handled through workspaces, which allow independent sessions, reduced resource contention and increased parallelism. Control of the simulation service operates through a master simulation service using the SOAP protocol and partitions the model execution through XML based messages. Message interaction between the components is handled by Linux message queues for mutual exclusion locking and some global data sharing among processes running on the same machine. Interfacing the Axis SOAP engine to the Linux message queues is the Java Native Interface (JNI) through a C++ library wrapper. Multi-threaded implementation of the DCD++ had increased system performance by having a client response thread and logging session thread for the WS component and simulation execution engine thread and an event queue thread for the simulation component [Mad06].

Thus, DCD++ in its design provides the following capabilities:

a. Session Management: user authentication and session administration;

b. Configuration: control of the MA extension file, DEVS model, event file, support file, execution time, and parsing info;
c. Monitoring and Control: control and monitoring of the session simulation process and state at runtime and external runtime event injection; and
d. Logging and Data Retrieving: user access to the generated DEVS log file, debugging information, session operational logs, and external environmental event output files [Zap11].

2.1.4 REpresentational State Transfer

Roy Fielding in his doctoral dissertation [Fie00] produced a separation of concerns of user interface and data storage to demonstrate portability and scalability for the evolution of components. He derived the REST as a constraint or distinguishing architectural boundary between components. The constraints for REST as described as follows:

a. statelessness: communication must be stateless where session state is to be kept on the client to improve visibility, reliability and scalability;
b. cache: data responses to be labeled cacheable or not to improve network efficiency and the average latency;
c. uniform interface: distinguish REST from other network based styles so that implementations are decoupled from services;
d. layered system: hierarchical layering to create a bound on system complexity and promote substrate independence where the physical construction of the server is fundamentally irrelevant. The benefit derived is encapsulation and enhanced scalability; and
e. code-on-demand: an optional constraint to allow client functionally extensions by reducing the number of pre-implemented features.

REST is a style of software architecture for distributed systems. It is a design pattern for creating WS where every distinguishable entity or value is a resource. A resource may be a web site, a Hyper-Text Markup Language (HTML) page, an XML document, a WS, or a

physical device for example. Every resource is uniquely identified by a URI as opinionated by Tim Berners-Lee's Axiom 0 [Ber96].

A design structure that is compliant with the REST style of architecture is said to be RESTful. A RESTful protocol uses the true potential of HTTP/1.1 and is oriented around verbs and resources. Such verbs are specified as GET, POST and HEAD for HTTP/1.0 and OPTIONS, PUT, DELETE, TRACE and CONNECT for HTTP/1.1. These verbs are then applied to the resources where acceptance of data is filtered and controlled by the HTTP headers. The client interacts needs no prior knowledge of the methods or how to interact with any particular WS beyond the knowledge of the URIs. All future actions of the client are discovered by the user within a resource representations returned from the server. A client transitions through application state rather than the server maintaining the state of the client.

Thus, to distinguish a user's perceptive for a RESTful approach, the resources may be accessible as follows:

a. ../account
b. ../account/xyz

To create a new user, a POST request is send to the URI with the appropriate data for an account creation. To retrieve data, a GET request is sent to the appropriate URI (../account/xyz). A PUT request with data would edit this resource while a DELETE request would delete the account. The main idea is that modification of a resource is never performed by a GET request. REST is an alternative to SOAP-based data exchange.

Conversely, to distinguish a non-RESTful approach, the resources may be accessible as follows:

a. ../account_create
b. ../account?id=xyz
c. ../account_modify?id=xyz
d. ../account_delete?id=xyz

By sending a GET request to a non-RESTful service, modification of data is permitted. GET requests can supply data in the form of parameters encoded in the URI by means of the query string or as cookies in the cookie request header. Using a web form for the above example, "../account?id=xyz", the "?" is defined as a separator and the query string "id=xyz" may cause a server to run a program, passing the query string unchanged to the program. A cookie request header is a HTTP cookie where the client transmits information to the server. An example would be "Set-Cookie: UserID=xyz; Max-Age=3600; Version=1" and as can be seen, data input is entered to the server without a POST or PUT operation.

2.1.5 RESTful-CD++

RESTful-CD++ is a RESTful HTTP server and DCD++ simulation [Alz11]. The application, hosted on a server, is commonly referred to as the RISE middleware. It is an REST implementation as defined by Roy Fielding, and provides its resources API based on URIs. Specifically, RISE enables distributed system interaction with its users by applying resource templates to provide uniform simulation environments and results.

RISE operates as a URI resource where users model simulation events by accessing the resources provided by RISE. To generate a CD++ simulation users have access to the following URIs [Wan12]:

a. by accessing the framework URI, users can submit their respective configurations and files for simulation by the CD++ simulation engine;
b. by accessing the framework/simulation URI, users can manipulate active simulations as this resource is a wrapper for the CD++ simulation engine; and
c. upon completion of the simulation, users can access the framework/results URI to retrieve the simulation results that is stored as compressed file.

2.2 Evolution of Distributed Simulation

Distributed simulation has come a long way since its first inception as a prototype research system to consider the viability of creating a real-time distributed simulator. The SIMulation NETworking (SIMNET) was the resulting application and grandfathered many

of today's dynamic implementations such as Z-Buffering and network protocols [Smi95]. The use of SIMNET during the "Gulf War" by the Americans demonstrated the success of real-time interactive networked cooperative virtual simulation. Advances in technology led to follow-on protocols such as DIS, HLA and the WebLVC protocol.

2.2.1 Distributed Interactive Simulation Protocol

DIS and the Aggregate Level Simulation Protocol (ALSP) replaced SIMNET during the 1990's. Its purpose was to allow dissimilar autonomous simulation nodes to interoperate in real-time, interactive, distributed simulations [Smi95]. At each simulation node, the simulation would maintain and convey the state of one or more simulation entities by exchanging standard DIS Protocol Data Units (PDUs) in a timely, succinct and consistent fashion.

The PDUs are documented as a unique numerical identifier of each simulated entity, presenting the spawning site, host, and entity numbers. Each entity, as represented by the PDUs, was able to convey its state, for example, displaying smoke, because of the enumeration and its bit-coded values [Smi95]. The communication medium for the PDUs was via User Datagram Protocol/Internet Protocol (UDP/IP) while the DIS Network Manager, which allowed multiple applications on a single host platform to operate and transmit their PDUs, used the Transport Control Protocol/Internet Protocol (TCP/IP) to exchange information. The DIS Network Manager acts as a smart proxy to queue UDP packets from a single port, filters them by type and re-transmits them via the TCP to its

client applications operating on the single host computer as shown in

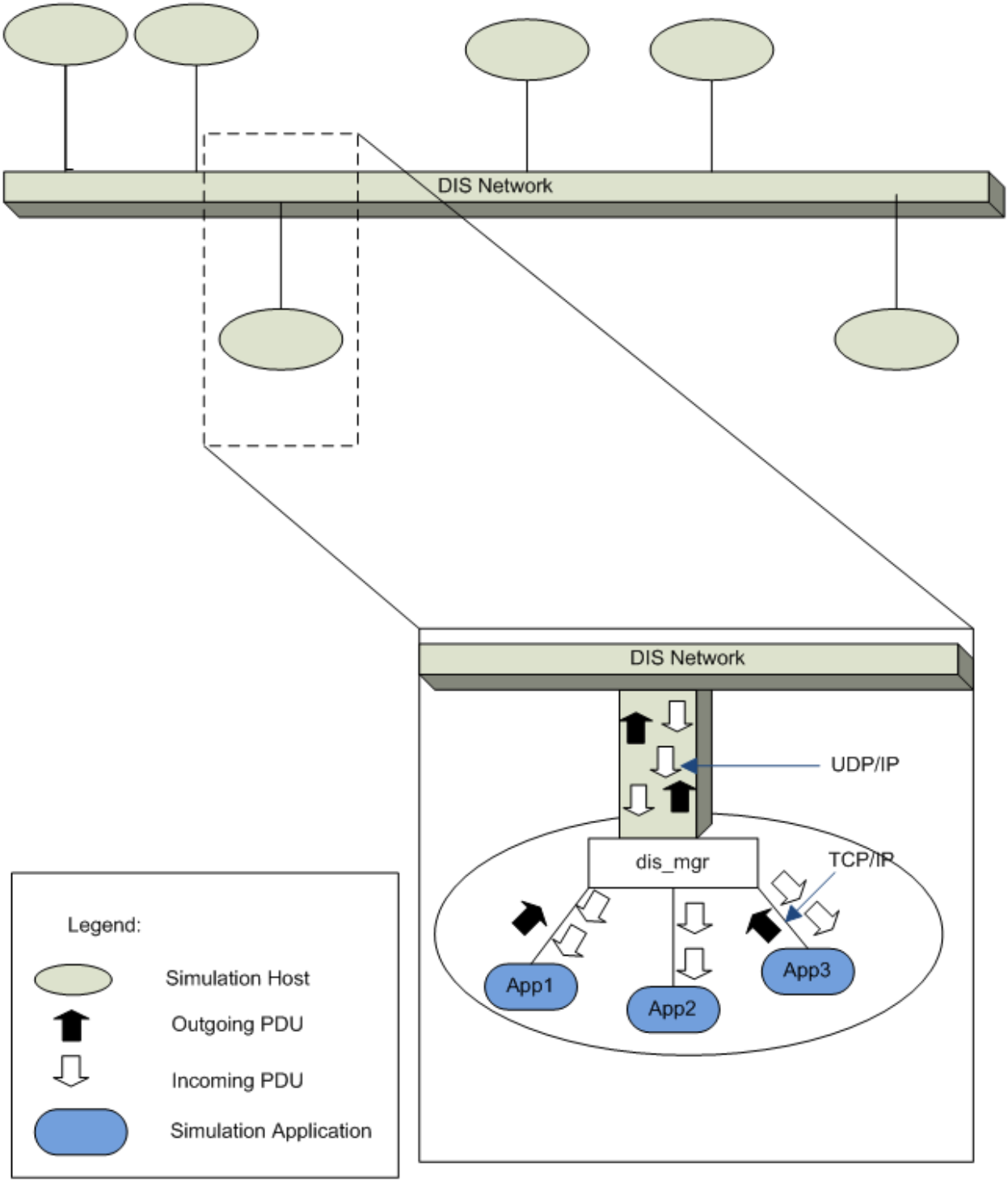

Figure 9 [Smi95].

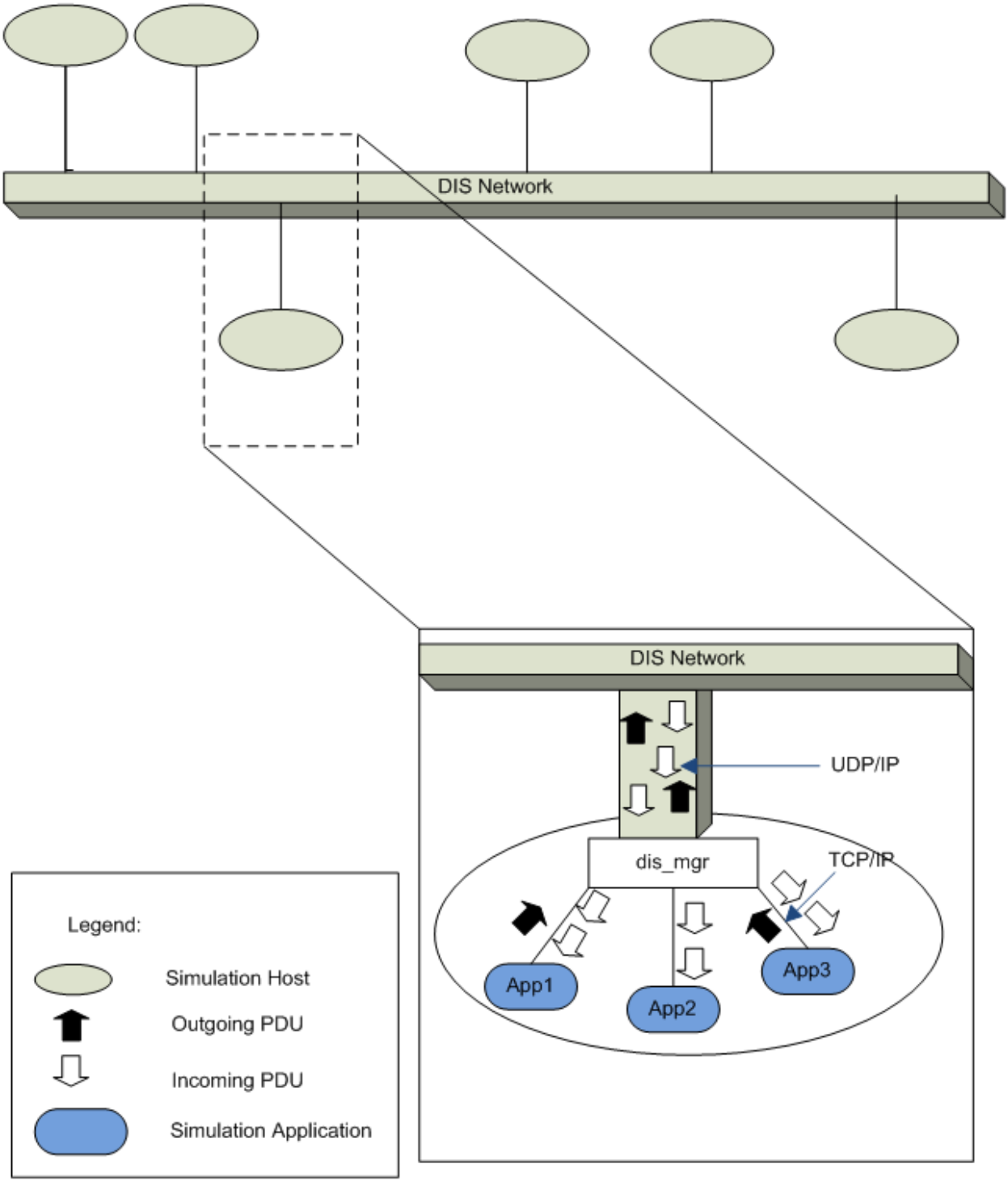

Figure 9. DIS Network Layout Connection.

2.2.2 High Level Architecture

HLA is a general purpose architecture framework for distributed computer simulations for reuse and interoperations of simulations. It was originally developed by the United States Defense M&S Office for the United States Department of Defense (DoD). It is now published by the Institute of Electrical and Electronic Engineers (IEEE) as a 1516 standard though DoD had released a complete 1.3 standard as shown in Figure 10 [McF12][Nat12] which is a timeline of the significant events in the lifecycle history of the standard of HLA. HLA is presently at version 1516-2010 or more commonly referred to as HLA 1516e.

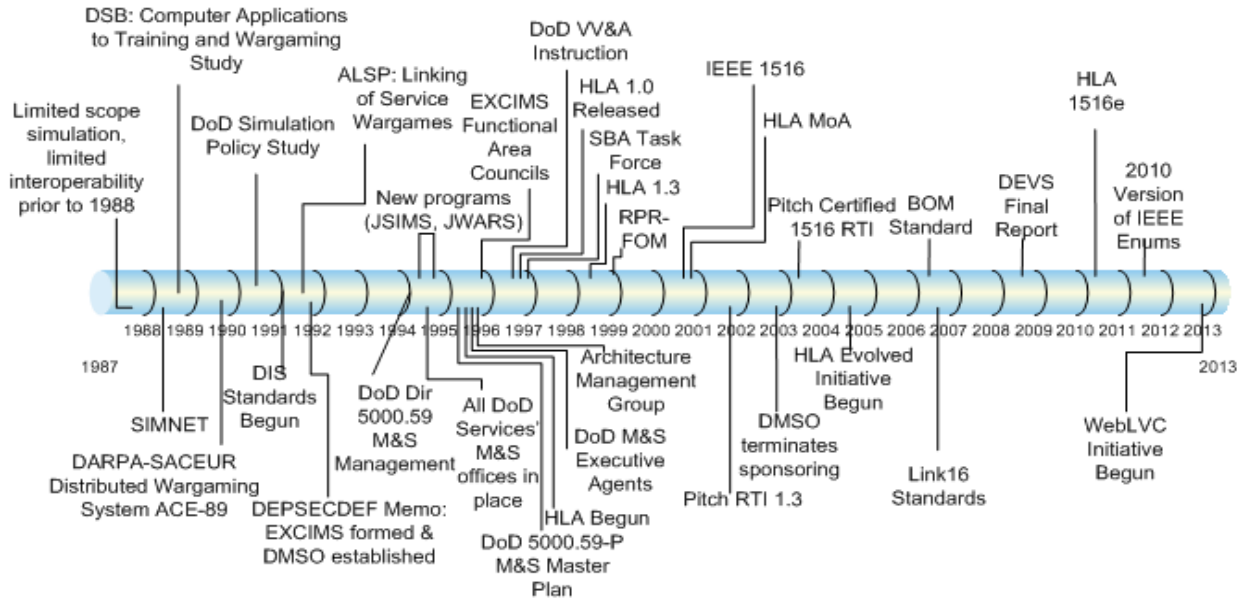

Figure 10. Distributed Simulation Timeline.

As many large and complex system simulations involve individual heterogeneous component simulations, HLA addressed the need for interoperability between new and existing simulations. HLA builds upon the DIS effort by the DIS Steering Committee in 1994 and the ALSP [Wil94]. Thus, the goal of the interfacing of the component simulations would be to provide reusability and interoperability of the aggregate component simulations. As system simulations were developed on dissimilar hardware and software platforms, DoD

required a common technical architecture which allowed the combination of the diverse component simulations for M&S.

HLA is still an evolving technology and is based on the advances made by the DIS. DIS specified the standard and layout of data transmission on a network, and was implemented using best-effort networking protocols such as UDP and broadcasting without a central coordinating server. Each simulation then had its entities provide a heartbeat to notify other entities of its continued existence. A benefit of the heartbeat is that individual component simulations could join or leave the aggregate simulation when desired and this is continued in HLA through its Federation Management runtime service.

Leveraging and improving the standard and data standards of DIS, HLA continues to allow component simulations to join or leave the aggregate simulation as desired but it requires a central manager called the RTI that receives data from the component simulations and sends the data to other component simulations through callback methods.

This RTI is the federation's simulation backplane and is responsible for federation set-up and retirement, synchronization, message ordering and data distribution. The component simulations are called federates and the aggregate simulation is called a federation. The rules or data format that the federates interact with the federation and the RTI is defined in the FOM [McF12]. The HLA standards dictate how federates communicate through their rules but do not dictate what they communicate. Rather, the FOM dictates what data is being exchanged in a particular HLA federation. Thus, each federation can be distinct for the data exchanged. Similarly, a federation of federates could be further classified as a federate of a "confederation". The term "confederation" is used only as identification means and is not syntactically correct but is entering mainstream use.

The Interface Specification (I/F Spec) [AEG07] as represented in Figure 11 that defines how HLA compliant simulators interact with the RTI and is composed of several services categories as follows:

a. federation management – controlling and destruction of the federation;

b. data distribution management - allows large federations to improve the efficiency of the distributed data by reducing the amount of data communicated by the federates by supporting efficient routing of data;

c. declaration management - provides the means to establish their intent to produce or consume object attributes and interactions from other federates through the RTI and describes the RTI subscription service;

d. object management - federates can create or delete object instances and produce or consume certain objects or interaction data;

e. ownership management - ability to transfer ownership attributes and the right to modify the value of these attributes of the entity where control of an entity is transferred from the management of one federate to another;

f. time management - allows the advancing of logical time; and

g. support services.

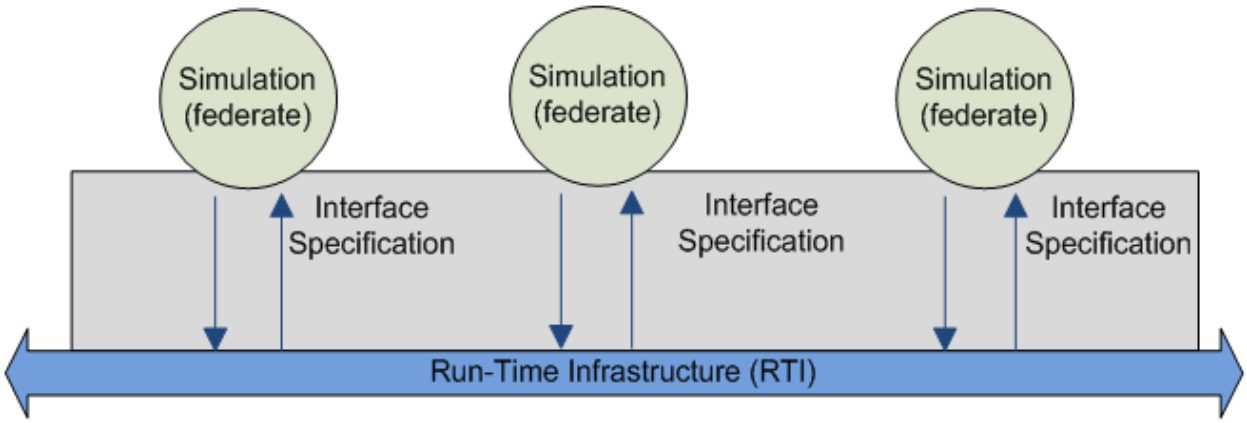

Figure 11. Federation Schematic

In HLA, the blueprint for the federation are found in the FOM which specifies the major features and relationships of the federation such as the object classes and interaction classes with their associated attributes and parameters. From the view of the federate, the Simulation Object Model (SOM) provides the blueprint of the federate's features and relationships of the object classes and interaction classes with their associated attributes and parameters [AEG07].

As the object classes and relationships have patterns of interplay, Base Object Models (BOMs) describe the essential capabilities and features needed for the conceptual level for a federate or federation. [Sis06]. Thus, the BOMs provide the component-based means for composability by describing the event type element relationship. Such events are used to represent and carry out pattern action as the directed exchange of information as triggers or undirected exchange of information as messages.

The RTI provides an efficient inter-federate communications software layer by providing common services to the federates by separating simulations concerns from communication concerns. With the introduction of the RTI and its API for communications, the federates are now language and platform independent. However, the RTI is vendor specific, such that federates cannot communicate with the federation unless the same vendor specific RTI is used. Thus, a confederation would require the same RTI. The RTI acts as intermediary software that provides common services to the federates. The communication transferred between federates in a federation or between federations in a confederation is through the RTI. To create such a bi-directional channel communication channel, each federate contains an RTI ambassador and a federate ambassador where the RTI ambassador handles all information passed from the federate to the RTI as an update. Each update on the RTI results in a callback message called a reflect message and is handled by the federate ambassador to invoke the appropriate code in the federate. This is illustrated in Figure 12 [AEG07].

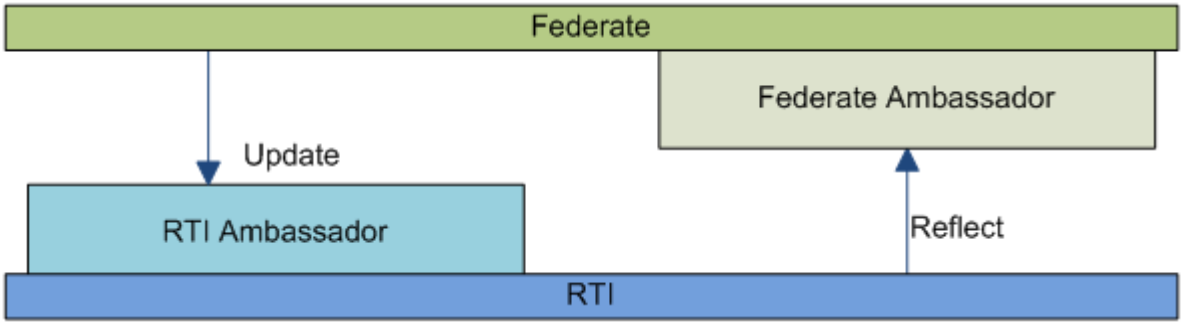

Figure 12. Federate / RTI Invoked Services.

Federation execution [AEG07] can be described under all the management aspects. Typically, the following is the timeline of a federation execution:

a. initialize federation:
    1. create federation – federation management, and
    2. join federation – federation management;
b. declare objects of common interest among federates:
    1. publish object class – declaration management, and
    2. subscribe to object class attribute – declaration management;
c. exchange information:
    1. update or reflect on attribute values – object management,
    2. send or receive interactions – object management,
    3. request or grant time advance – time management,
    4. attribute ownership assumption – ownership management, and
    5. modify region – data distribution management;
d. terminate execution:
    1. resign federation execution – federation management, and
    2. destroy federation execution – federation management.

As with all software products, refactoring of source code may be required when the standard or API changes [AEG12]. Different versions of HLA have different federates with different HLA version compliant RTIs. As refactoring these complex systems may not be possible because of man-hours of intensive work, fiscally costly, or the user does not have the source code to migrate the Commercial Off The Shelf (COTS) federate from one standard to another, or legacy federates will not be maintained, vendor specific adapters have been created to broker the formatting of data and provide API calls and call-backs for a federate. These will, however, cause technical problems and limit the HLA 1516 federates because the HLA 1.3 federate will not take advantage of the HLA 1516 standardized data encoding. An example of such an adapter is Pitch's interoperability solution "1516 Adapter" for HLA 1.3 federates [AEG12].

2.4 Modern Web Technology

Exciting advances in client web browser technology has allowed the high-fidelity visualization of streaming video. The combination of HTML5 canvas elements, WebSockets, web workers, and Base64 encoding has provided this user's capability to view image data in a streaming video-like quality.

2.4.1 HyperText Markup Language 5

The WebLVC protocol and server would not have been possible without the advances in web technology. HTML5 is the newest version of the markup language used for structuring and presenting content for the World Wide Web. This revision is still under development though most modern browsers are fully compatible of supporting the version. HTML5 was designed to provide a single markup language between HTML and eXtensible HyperText Markup Language (XHTML) while also implementing other new elements such as audio, video and canvas elements to name a few. These new elements are designed to handle graphical multimedia rich environments without having to resort to third party plug-ins.

The HTML5 canvas element allows the client web browser to inherently manipulate, construct and layer image data [Wes11]. The canvas element provides a destination for rendering in web pages and allows for the Graphics Processor Unit (GPU) accelerated usage as a Document Object Model (DOM) interface. In order to prevent flickering when the image is redrawn, a render loop is created allowing the web browser to double buffer the rendering surface creating the ability to overlay the new image directly over the old one without the need for third party proprietary plug-ins.

WebGL which allows 2D and 3D graphics is not a W3C standard but a JavaScript API by the Khronos Group. However, Microsoft has stated that "browser support for WebGL directly exposes hardware functionality to the web in a way that we consider to be overly permissive" [Mic11]. Their stance is that security attacks are capable of reading private information from the memory of the card or perform denial of service attacks. Only Firefox,

Chrome, Safari and Opera support WebGL while Microsoft and mobile browsers do not. All the above browsers though, support the W3C canvas element and TypedArrays.

2.4.2 WebSockets

The transportation of the JavaScript Object Notation (JSON) objects can be passed via a bidirectional communication channel called WebSockets. In a way, they are a combination of UDP and TCP in that they pass messages like UDP but have the reliability of TCP. With a combination of the two protocols, a client is able to create an asynchronous full-duplex channel to the host server. This communication allows the client to send data instantly to the server and have the server communicate to the client at the same time while the connection is open.

WebSockets, besides defining a new protocol for the transference of data, also provides a method for creating secure connections [Wes11]. Similar to normal asynchronous calls like TCP, where the protocol is optimized for accurate document delivery rather than timely delivery in that all bytes received will be identical to all bytes sent and in the correct order. WebSockets do not have the problem of TCP as TCP incurs relatively long delays while waiting for out-of-order messages or retransmissions of lost messages using its positive acknowledgement technique. This TCP technique requires the receiver and sender to send an acknowledgement message each time it receives data segments, preventing the streaming of data. Critical to the accurate delivery technique, in TCP, the sender is required to keep a timer on the transmitted packet so that if an acknowledgment message is missed, the timer will expire and the transmitted data is resent. Once the full data is transmitted, the receiver no longer talks to the sender.

WebSockets, in comparison, use an accurate and secure communication technique directly from the receiver to the sender and streams the data along this communication channel. Rather than the receiver asking if there are updates from sender through a poll mechanism such as keeping the communication channel open and inquiring about updates, WebSockets keep the channel open and request that the sender informs the receiver if

there is an updates without a polling mechanism. When the sender has data, it informs the receiver and sends the data.

Request For Comments (RFC) 6455 [IET11], the WebSocket protocol, ensured that HTTP web based clients and WebSocket based clients can operate on the same port. With a common port, the WebSocket client establishes a connection by sending out a request to connect and if the server desires the connection it will send out a response accepting the new connection. Upon establishing the handshake connection, both the client and server upgrade from a HTTP based protocol to a WebSocket based protocol. The origin header of the client specifies the domain of the client giving the server the decision to accept different origins. The reduced bandwidth consumption and increased message passing ability greatly reduce the computation and rendering limitations allowing faster, more scalable and more robust high performance real time applications.

Comparing TCP to WebSockets, it can be calculated that the network throughput overhead is significant such as using a JavaScript stock ticker application where data is updated once per second [Lub12]. Figure 13 [Lub12] details the typical HTTP request header while Figure 14 [Lub12] details the HTTP response header.

```
GET /PollingStock//PollingStock HTTP/1.1
 Host: localhost:8080
 User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.1.5)
 Gecko/20091102 Firefox/3.5.5
 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
 Accept-Language: en-us
 Accept-Encoding: gzip,deflate
 Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
 Keep-Alive: 300
 Connection: keep-alive
 Referer: http://www.example.com/PollingStock/
 Cookie: showInheritedConstant=false;
showInheritedProtectedConstant=false;
 showInheritedProperty=false;
showInheritedProtectedProperty=false;
 showInheritedMethod=false; showInheritedProtectedMethod=false;
 showInheritedEvent=false; showInheritedStyle=false;
```

showInheritedEffect=false

Figure 13. HTTP Request Header.

HTTP/1.x 200 OK
X-Powered-By: Servlet/2.5
Server: Sun Java System Application Server 9.1_02
Content-Type: text/html;charset=UTF-8
Content-Length: 21
Date: Sat, 07 Nov 2009 00:32:46 GMT

Figure 14. HTTP Response Header.

The HTTP request and response headers total 871 bytes, without accounting for the data being transmitted. In a polling version of the application, Lubbers suggests using a use case scenario which has different numbers of users requesting data for a comparison between TCP and WebSockets as shown in Table 1 [Lub12]. In this regard, he calculates the required network throughput for the different use cases for each protocol. In Figure 15 [Lub12], he graphically summarizes the use cases. It is readily apparent in the presented data that if the clients number increases or the amount polling time decreases to 100ms, network throughput requirements is unsupportable for TCP while the WebSockets network throughput requirements remain significantly uncontested and very much attainable.

TCP	Use Case A	1,000 clients polling every second: Network throughput is (871 x 1,000) = 871,000 bytes = 6,968,000 bits per second (6.6 Mbps)
	Use Case B	10,000 clients polling every second: Network throughput is (871 x 10,000) = 8,710,000 bytes = 69,680,000 bits per second (66 Mbps)
	Use Case C	100,000 clients polling every 1 second: Network throughput is (871 x 100,000) = 87,100,000 bytes = 696,800,000 bits per second (665 Mbps)
WebSockets	Use Case A	1,000 clients receive 1 message per second: Network throughput is (2 x 1,000) = 2,000 bytes = 16,000 bits per second (0.015 Mbps)
	Use Case B	10,000 clients receive 1 message per second: Network throughput is (2 x 10,000) = 20,000 bytes = 160,000 bits per second (0.153 Mbps)
	Use Case C	100,000 clients receive 1 message per second: Network throughput is (2 x 100,000) = 200,000 bytes = 1,600,000 bits per second (1.526 Mbps)

Table 1. Network Throughput Requirement for TCP and WebSockets.

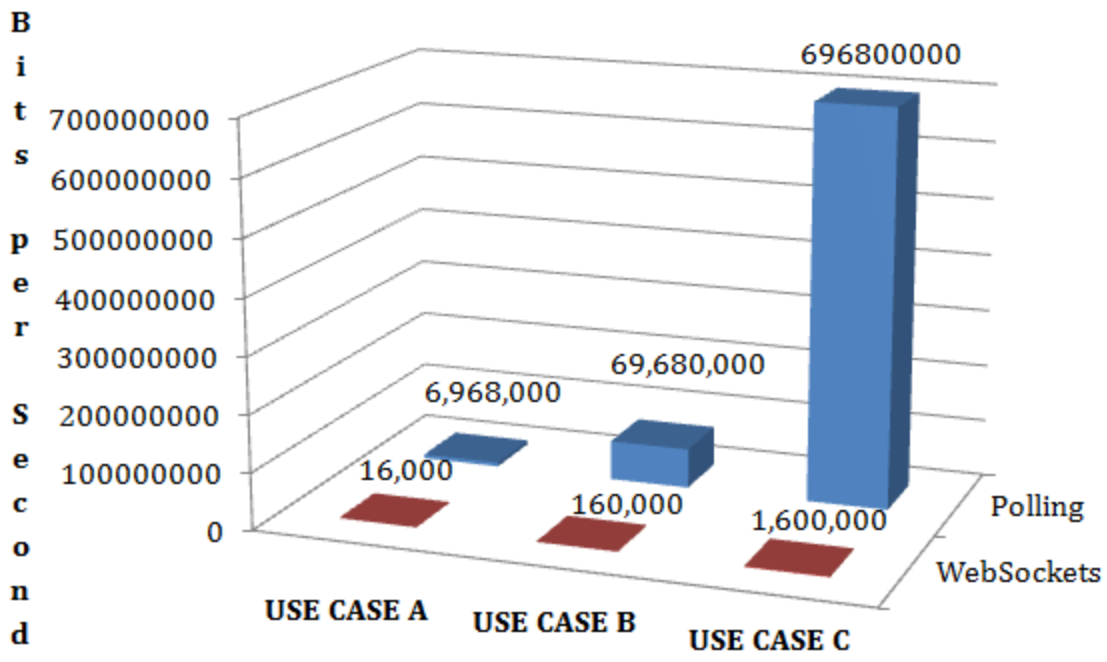

Figure 15. Comparison of the Network Throughput of TCP and WebSockets.

2.4.3 Web Workers

Simply put, web workers allow multi-threading in JavaScript. Though web workers are in its infancy, client applications are no longer single-threaded and have to rely on server generated state information. Processing can now be off-loaded back to the client, freeing the server from computational requirements and state changes. Originally, the main User Interface (UI) thread was the only thread that was allowed to exist and all data and events were sequentially executed. With the advent of web workers, multi-threaded client applications can be created where the main UI thread communicates to child threads via message channels and data is passed through the message commands and message data. Following on to this concept, the child threads can now spawn their own child threads to off-load computational processing. With this direction, JavaScript execution is moving further into the main stream of modern technology by being able to use multi-processor systems to augment its versatility.

Web workers can be defined in two ways – either as a dedicated worker or a shared worker. A dedicated worker can only be accessed by the script that created it while a shared worker can be accessed by any script controlled through multiple port access. Generally, according to W3C, web workers "are expected to be long lived, have a high start-up performance cost, and a high per-instance memory cost" [W3C12 - WW]. Web workers can load scripts or modules using the importScripts command, attach error handlers, and terminate themselves upon completion. Shared web workers, because of their multiple port communication, can close the port to prevent communication on that port.

However, web workers have been designed to not be able to access the DOM because of thread-safety, the window object, the document object, or the parent object. It also does not have access to create and bind a WebSocket. It does, however, have access to the navigator object, a read-only capability of the location object, XMLHttpRequest, timers, and the application cache. The Web worker can import external scripts and spawn other web workers.

2.4.4 Base64 Encoding

Though not new, Base64 encoding is used to store and transfer binary data over mediums that are designed for textual data. Thus, data remains intact without modification during transport and to prevent collision problems. Without the Base64 encoding, the image data being transmitted would interfere with the WebSocket protocol [Wes11] if the server only has textual output. Base64 encoding operates by representing the binary data in an American Standard Code for Information Interchange (ASCII) string by translating the data into a radix-64 representation. As the JSON format does not natively support binary data, Base64 is strictly an encoding scheme used to represent binary data in an ASCII format allowing the transfer of data over a medium that is designed strictly for textual data. Originally, though the web browsers inherently allow Base 64 encoding / decoding, WebSockets were incapable of sending binary data such as a Binary Large Object (BLOB). However, modern browsers now allow data types such as ArrayBuffer for binary data.

Base64 encoding is used also in the handshake connection of the client and server for WebSockets. The use of Base64 encoding on the Sec-WebSocket-Key and the Sec-WebSocket-Accept ensures that the client understands the data of the WebSocket. Even if there is the chance that the client and server may interpret the same thing differently, by having an agreed format, the data will not be interpreted incorrectly. This is created to prevent a cross-protocol attack; but in order to pass data, the Base64 option must be set correctly.

2.4.5 Asynchronous JavaScript and XML

Asynchronous JavaScript and XML (AJAX) is a combination of technologies available since 1999, with the introduction of the XMLHttpRequest component in Internet Explorer 5.0. The main characteristic of AJAX is its asynchronous nature allowing the transmission and reception of data from a server without having to manually refresh the page. Once the client receives the response from the server, JavaScript can modify the contents of the displayed page allowing the user to continue the interacting with the page without having to worry how the server and client are interacting in the background.

Thus AJAX has become the fundamental technique for creating dynamic usable web applications. Dojo simplifies these interactions by providing a simple unified portable API to wrap XMLHttpRequest functions, and to handle form manipulation with ease [Doj12].

2.4.6 W3C File API

The World Wide Web Consortium (W3C) File APIs of HTML5 allow JavaScript to access to files in the local file system by representing the file objects in web applications [W3C12 - FA]. Previously in HTML – version 4, third party applications such as Silverlight, Flash or a Java application with an ActiveX component were the only methods for older browsers to allow access to the local file system or using only JavaScript and the input type element where the user uploads the file to a server for processing. In HTML5, the JavaScript API allows a standardized file capability directly from a user's selection of the file by use of the File API and the conversion of data contained in BLOBs.

With the advent of client mash-up services, higher generation mobile devices and the client controlling its state, it seems inevitable that the W3C would desire the ability for client applications to become robust and standardized without invalidating security policies. Directly, there are two types of file access – the File API allows read-only capability while the BLOB API allows full file system access. Presently, only the Chrome browser implements the allocation of dynamic memory for the BLOB API.

2.4.7 W3C Cross-Origin Resource Sharing

Security in client side programming languages such as JavaScript are important in order to prevent the loss of data, confidentiality or integrity by providing a strict separation between content provided by unrelated sites. The same origin policy uses a combination of protocol, host and port number to provide limitations to the access of information and resources from other sites or even the same domain with different ports [W3C12 -CORS].

CORS is a draft standardization from the W3C relaxing the same origin policy [W3C12 -CORS]. Though there are holes within the same security concept such as message passing techniques and frame or window domain setting techniques, CORS allows relaxation by extending the HTTP request header with a new origin field and the HTTP response header with a new Access-Control-Allow-Origin field. HTTP or HyperText Transfer Protocol Secure (HTTPS) requests using XMLHttpRequest for asynchronous communication and the CORS specification allow a client to directly request a WS and load the response data directly back into the script. Without the CORS specification, XMLHttpRequest is limited to the same origin policy.

With the advent of Web 2.0, mash-ups have allowed the mixing and matching of diverse proprietary APIs to create new services. With the focus of the WWW turning to RESTful WS based on the strength of the client side platforms, mash-ups of services are being offered under the guise of singleton applications such as visualizations of mappings, photography

services or as enterprise services providing a central point of contact for internal data through external data WS.

2.4.8 Evolution of Modeling and Simulation

The advancement of the DEVS formalism has created various different aspects of research [Kha05]. Presently, researchers have examined and created different means of creating parallel and distributed DEVS environments in terms of the middleware tools to implement the DEVS simulator and the functionality it offers. Researchers have suggested grid environments of homogenous platforms to provide the dynamic aspect of producing an M&S environment through the use of grid middleware for resource allocation and management, authentication, authorization and communication among its simulation nodes [Kha05]. Others have focused on the design aspects of parallelism to provide acceleration of the simulation engines through the use of optimistic simulation algorithms to advance their clocks independently rather than in a synchronized manner [Kha05].

Traditional distributed simulations, primarily developed for military applications, are entering into the domain of the civilian world through the use of COTS software, which allow simulations to interact through the communication of data and synchronization of interactions. Through the use of a publish and subscription model, both the distributed simulation's objects and its interactions can be advanced to other compliant distributed simulations. Research has been developed to advance the specification to the level of a soap based WS [Mol12]. Research has also begun on the WebLVC initiative [MAK12].

Chapter 3: Trends in the Implementation of Distributed Simulation Environments

In this chapter, we introduce an overview of some of the major reasons to provide a web based distributed simulation and to provide a web based DEVS environment. Then, we introduce differences in the implementations of the services and the technique that we propose in this dissertation in terms of the methodology we followed, the middleware used for the implementation as a proof of concept and the advantages of service it offers when operating as mash-up of services.

3.1 Web Service-Enabled CD++

3.1.1 SOAP Based DCD++

Advancements were applied to the DEVS M&S environment in order to provide the CD++ toolkit functionality in order to execute and retrieve simulation results through the technology of WS. The main component that allows the DCD++ WS is the wrapper which is composed of a WS component, written in Java, and a Simulation component, written in C++ [Zap11][Kha05]. The WS component services user authentication, session management and parsing simulation requests and is deployed as an Axis SOAP engine running via an Apache Tomcat application server [Zap11]. The Simulation component accesses and manipulates the internal objects and data structures in the simulation engine.

Client session interaction is handled through workspaces, which allow independent sessions, reduced resource contention and increased parallelism. Message interaction between the components is handled by Linux message queues for mutual exclusion locking and some global data sharing among processes running on the same machine. Interfacing the Axis SOAP engine to the Linux message queues is the JNI through a C++ library wrapper. Multi-threaded implementation of the DCD++ increases system performance by having a client response thread and logging session thread. These threads are the WS component and simulation execution engine thread and an event queue thread for the simulation component [Zap11][Kha05].

DCD++, using the SOAP protocol, enabled a distributed version of CD++ to execute complex models on multiple machines. With SOAP retrieval, one can send a complex request, in structured XML format, with long and complex conditions. Its model, however, continued a tight coupling between different services as it is a protocol rather than an architectural style. As the use of SOAP benefits contractual WS, it is based on the message concept and has its integrity and confidentiality designed as token formats to provide end-to-end security. Its services can be easily consumed by client applications, has rigid type checking as it adheres to a contract, and has rich but third party development tools. Because of the contractual nature of SOAP, developers would require specific knowledge of the XML specification, and usually require a SOAP toolkit to form requests and parse the results, as SOAP requires an XML wrapper around every request and response. The strong typing of SOAP has been proposed as necessary feature of distributed simulations. However, in practice the requesting application and the WS know the data types ahead of time showing that strong typing is actually unwarranted.

DCD++ is a SOAP based WS that has not been modified to provide a traditional distributed simulation interface. In order to create a FOM for DCD++, the exact object and interactions would have to be developed and the code refactored in order to provide an interface for a distributed simulation. As DCD++ is web-based, the logical interface would be to publish and subscribe to a HLA 1516e SOAP based service; however, this is not the only alternative.

3.1.2 RESTful-CD++ Interoperability Simulation Environment

Differing from SOAP, the use of REST supports that the present existing principles and protocols can create robust WS. These principles and protocols, namely the HTTP and HTTPS protocol, are mapped to URI specific resources. Strictly speaking, REST requires that access to resources is through the GET, PUT, POST and DELETE methods. This means that the client side application is thinner in that a web browser can now be used as the client application rather than a heavy client side application as required for SOAP.

The key goals of REST, as stated by Dr Fielding in his dissertation, is the scalability of component interactions, its generality of interfaces, its independent deployment of components and its intermediary components to reduce latency delays, compel security implementations and encapsulate legacy systems [Fie00]. Based on this architectural design philosophy, RISE was developed as a URI-oriented architecture where service URIs are structured in a hierarchal tree and client applications such as browsers, manipulate the resources of the server through the URIs. RISE uses URI templates to expose its common resource and provides administrative services, structural and extendable framework for simulation services as seen in Figure 16 [Wan12]. Since the URIs names the resources, RISE encapsulates technical functionality and provides a platform that is independent of formalism or tools.

RISE is a platform and service that was implemented to provide server resource availability for client applications. Specifically, RISE was created as a middleware resource for a RESTful implementation of the distributed CD++. CD++ is a toolkit for Discrete-Event M&S and is based on DEVS. The design modeled into the RISE middleware was to provide transparent sharing of computing power, data models, and heterogeneous environments on a global scale. RISE provides a lightweight approach to WS. It hides internal software implementation as compared to the SOAP based WS, which rely on Remote Procedural Calls (RPCs). These RPCs belie and exasperate the internal structure dynamics of the application as procedural parameters and constrain the correct marshalling of data is by its internal data structures.

Both DCD++ and RISE rely on the CD++ toolkit to generate output for the user. Simulation results are stored within a textual log file that contains the line by line event status of the CD++ simulation execution. A part of the CD++ toolkit is a drawlog executable that produces textual ASCII diagrams to represent the data for visual verification. Research has been forwarded for third party visualization tools but user input is required to directly manipulate the output log into a format acceptable for third party rendering tools. However, it must be noted that each simulation output log has to be manually manipulated and directly coded in order to display the simulation results in a third party viewer.

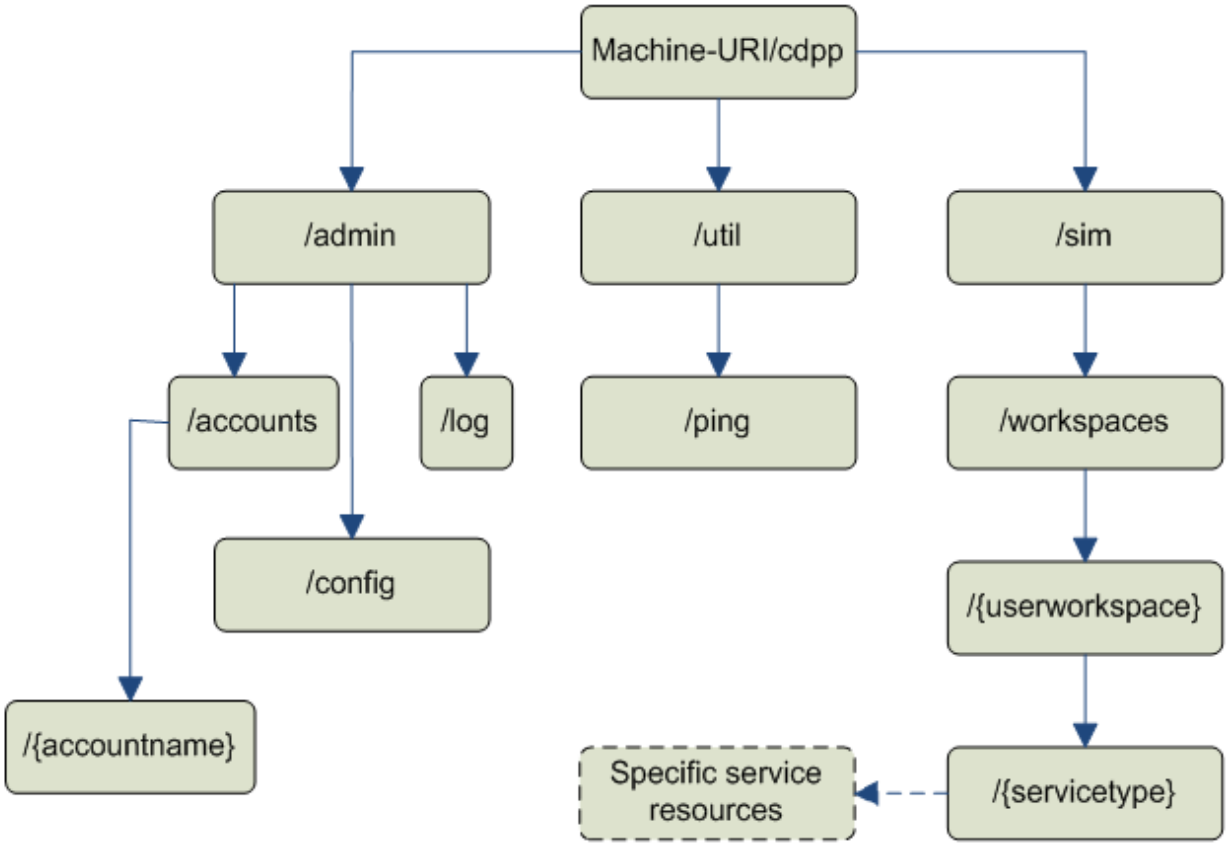

Figure 16. The RESTful-CD++ Common Resource Structure.

RISE, as a REST based WS, requires that it remain stateless and the client application control the state changes. It, therefore, presents the resources as URIs but does not contain a distributed simulation interface URI. Thus, simulation results of modeling world events are confined to researchers knowledgeable of DEVS [DEV08] and the number robust client applications accessing the simulation data are limited.

3.2 Agent-based Modeling

Distributed simulation produces a simulation as an agent-based model simulating the actions and interactions of autonomous active entities or environments in a network. The entities are active in that they may actively make decisions, retain memory of past and situations and decisions, and exhibit learning [Dod99]. The environment parameters, as used in DIS [DIS11], are usually static in that a generic status is provided to represent a change in the environment. Craters, smoke, clouds and flashes are observable environment changes that are transmitted through the network to provide a low-level fidelity within the

virtual world of the event situation. Furthering this limitation is the fact that the environment entities are created either statically upon commencement of the simulation or dynamically as the simulation progresses. The dynamically environmental changes are simplistic in nature as they are represented by an instantaneous creation at some fixed point in time and fixed in size through its data structure, though the event size of such an entity can grow only symmetrically. Examples of this dynamic environmental entity creation may be a crater or a fixed point, or a cloud of smoke surrounding another entity such as a truck or building. The visualization of the entity is a constant but the attributes of the entity are modified to denote an increase in radius.

However, as realism is evaluated by the user's perception of the visual simulation, these dynamically created environmental entities only provide low-level simulation results. Visually modelling a simple fire is Central Processing Unit (CPU) intensive and continually changing because of the interaction with the fuel, air and heat of the burning and surrounding materials. Though preferable to provide high fidelity, modeling such dynamic behaviour is contrary presently to the goal of providing a simulation that enhances and trains members for force generation. However, if a known area is pre-processed based on the materials present, and weather conditions, the environment entity such as a fire could be realistically interfaced to provide valuable training aid allowing the force commander to decisively make decisions based on the rate of consumption of the fire, the heat value of the fire, whether the fire will affect surrounding areas, whether the advance of the infantry must be altered or whether to use the environment entity as camouflage. Similarly, the destruction of a dam can affect force generation with the advancement of the water and the natural effects of flooding on the troops and surface conditions. Since a virtual simulation is usually not executed in real time but rather in simulation time, the state change on the flooding environmental entity is a major consideration in view of the devastation created by Hurricane Katrina in New Orleans when the levies collapsed or the Fukushima reactor meltdown caused by the 2011 tsunami. Examples of the realism as stated above provide the need and desire to promote a dynamic and complex interaction rather than a statically and simplistic simulation.

3.3 High-Fidelity Modeling Environment

Modeling dynamic terrain is crucial to the fidelity of any simulation in order to provide a realistic synthetic battlefield environment. By providing dynamic and interactive terrain changes, force commanders can be provided with decision-making opportunities and can decisively evaluate changing tactics based on the environment. Terrain changes such as a flooding river, forest fires caused by the weapon discharges or craters caused by the controlled tactical release of land modifying armament can impede or hinder both the enemy and the friendly advancement force generation. In a war stricken area with heavy fires or detonations, armament barrage strikes can destroy protective berms and allow armoured fighting vehicles to cross rather the just the infantry. Thus, dynamic and realistic modifications to the environment such as the terrain illustrate the importance within virtual simulations.

Presently, SISO-REF-010 [DIS11], the Enumeration and Bit Encoded Values for Use with Protocols for Distributed Interactive Simulation Applications, commonly referred to as DIS Enumerations, provide environment family PDUs which allow dynamic military concerned entities such as craters, bridges, and appearance attributes. However, in order to provide a non-militaristic component, the DIS Enumerations allow for a modifiable generic Environmental Process PDU to create entities such as points, lines, spheres, ellipsoids and others types. In distributed simulations, such as with HLA, HLA dictates the 'how' the federates exchange data, while it is the FOM that dictates what data is being exchanged in a federation [AEG07]. As federates must agree on a particular FOM in order to communicate their objects and interactions, SISO developed a Real-Time Platform-level Reference FOM (RPR-FOM) with the goal to provide an able interpretation of the concepts used in DIS for an HLA environment.

In order to provide dynamic environmental entities as an agent-based modeling, the paradigm of modeling requires a complex modeling environment outside of the structure of the agent-based model yet interfaced and providing input into the simulation. The

reason for a modular interfaced environment is based on the number of already developed simulations within the agent-based modeling simulation world. Refactoring and compliance to the standard described in distributed simulations would be a hindrance rather than a benefit.

3.4 WebLVC Protocol

Presently being developed and at its infancy stage, MÄk has taken the initiative to provide a suite of software applications that would allow web-based clients to interact with traditional modeling and simulation federations. Any of these federations may be using the DIS, HLA or TENA. Prototyping the protocol and submitting an initial draft to SISO for study, MÄk has provided the framework for a consensus-based interoperability standard using the JSON objects matched with the built in encodings for DIS and RPR-FOM semantics.

The WebLVC server permits the protocol to define a standard way of passing simulation data between a web-based client application and a WebLVC server while remaining independent of the protocol used within the federation. Hence, a web-based client application using the WebLVC server could participate in distributed simulation exercise such as a DIS exercise, an HLA federation, a TENA execution, or any other distributed simulation environment [MÄK12].

In promoting the WebLVC protocol, MÄk had developed prototype JavaScript implementation of 2D viewers using the Environmental Systems Research Institute Aeronautical Reconnaissance Coverage Geographic Information System (ESRI ARCGIS) street map, the ESRI ARCGIS Imagery World 2D map, the OpenLayer Open street map, a 3D viewer using Google Earth, a message inspector to inspect the JSON entity messages, a simulation manager, an Unmanned Aerial System (UAS) controller and an entity simulation prototype. Developed client applications could be hosted on their cloud-based test bed platform or run using a local copy of the WebLVC server which could be downloaded with a

license. A draft specification of the WebLVC protocol, version 0.1, could also be accessed at the SISO discussion forums [MÄK12].

3.5 Intermediary Mash-up of Services

In answering the question of why integrate a discrete event simulation with distributed simulation, Zeigler and et al created a HLA 1.0 compliant M&S environment as part of the Advance Simulation Technology Thrust (ASTT) for the Defense Advanced Research Projects Agency (DARPA) [Zei98]. In this regard, they were able to provide a mapping of the DEVS to an HLA federation (DEVS/HLA) specifically for a purser/evader federation. The federates were created based on a component model of an object and a DEVS atomic level model as a driver and interacted with the HLA 1.0 RTI. Though this demonstration validated the DEVS/HLA interface and provided "a test-bed to study generic architectures for predicative contract mechanisms" [Zei98], it did not produce a generic interface allowing multiple or different DEVS models to interact with the distributed simulations. Specifically, any DEVS modeller would have to declare the objects, attributes, interactions and parameters desired for the reflecting states of DEVS federates prior to participating in a distributed simulation.

Joint Mission Effectiveness Analysis Simulator for Utility, Research and Evaluation (MEASURE) [Zei99] was created after porting Pleiades, a DEVS-compliant simulation system, to execute on a DEVS/HLA environment as mentioned above. Led by Lockheed, and its subsidiaries, it attempted to study the mission effectiveness of systems within larger systems of system configurations. Because of the proprietary nature of the source code both from the subsidiaries and individual customer viewpoint, source code was not allowed to be shared within the corporation. It, instead, required the collaboration of the individual subsidiary developers to develop the appropriate FOM and SOM interfaces for the model components. Thus, each model component was required to be reprogrammed with the HLA interfaces prior to the establishment of a federation and the execution of the model component as a federate.

Significant efforts were made to interface DEVS to HLA but required to the refactoring of each the DEVS models into a configuration acceptable for the HLA version. As Cell-DEVS has been used to simulate many diverse and complex systems, for example, Madhoun and et al modelled battlefield scenarios [Mad05], each model would require extensive man-hours to facilitate the deployment of the DEVS models into a distributed simulation. We propose that the models remained unchanged and that a technique be implemented allowing an intermediary means to manipulate the model's data and publish the data to a distributed simulation environment. This technique would draw on the advances of the web technology, RISE, and the WebLVC protocol for communication into the distributed simulations.

By utilising the prototype WebLVC protocol, we propose to provide a novel means for a distributed simulation that interfaces with distributed simulations without requiring modification to the original source code and as an additional benefit, a visualization tool. In order to provide such a means, we propose to use and combine the data and functionality of RISE with the interface functionality of the WebLVC server. In joining these two components, a mash-up of services is created to generate a new service separate from the original intent of the objectives for CD++ and distributed simulations. As mentioned above, client applications such as a browser can access the RISE resources, provide inherent JSON parsers, and display real-time visual information through WebSockets. We propose using a browser as the means to provide the mash-up of services and become the flexible intermediary technique between RISE and distributed simulations.

Chapter 4: RISE-Distributed Simulation Technique

In distributed simulation environments, focus has been based on the ontology components in order to achieve simulation reuse of simulated systems such as functional-level behaviours, attributes and properties. Because of the complexity of distributed simulations, research has been extensive on creating visual coding tools that allow the simulation to present its data in the correct data types and formatting in accordance with the distributed simulation interoperability architectures and environments.

Access to distributed simulations has not been established by web-based architectures because of the limited capabilities of the thin client and latency of the data. However, with the combination of new technologies and standards, monolithic applications present within a distributed simulation can now be interfaced by the WebLVC server. The server stands connected to the distributed simulation similar to a DIS/HLA gateway but receives and transmits JSON data to web-based simulations through its WebSockets. Combining this technology and methodology, we propose to directly interface and integrate discrete event simulation data as presented on RISE into a distributed simulation regardless of the distributed simulation environment as shown in Figure 6.

Presently, the RESTful-CD++ data is housed on a simulation engine platform called RISE which interfaces with the user through HTML and XML messages. RISE is not capable of presenting data in a format acceptable for traditional distributed simulations. It has neither a distributed simulation interface nor a middleware component that could adapt multiple data models into a form understandable for distributed simulation. Since the DCD++ simulation engine can model and simulate a complex system under study, focus should be applied to expose these simulation results into distributed simulations. With the RISE simulation data present in a distributed simulation, training and force generation capability can be enhanced through a simulation simulating the actions and interactions of autonomous active entities or environments in a network.

The WebLVC server is designed to provide three interfaces allowing distributed simulations activity. It is presently capable of interfacing to DIS and HLA distributed simulations and to web based thin clients through the WebLVC protocol. It uses the MÄk WebLVC tool suite to provide a starting point for entity generation for thin clients. Still in its infancy, the WebLVC interface is the only interface to allow boundary information exchanges with client applications. Generation of single basic vehicle type entity information follows the DIS 1.0 specification, though modified from an enumeration type to a JSON format object. Rendering of the distributed simulation world are provided to allow rendering in both 2D and 3D planes using third party web accessible dynamic imagery.

In the context of our technique, a mash-up of services are introduced to serve six main purposes:

a. to expose a web based discrete event simulation to a distributed simulation;
b. to expose the RESTful-CD++ simulation results that is accessible through RISE, where user interaction is limited to directly accessing the RISE resource results file;
c. to manipulate the data into a format understandable to distributed simulations and specifically the WebLVC server;
d. to publish multiple distributed simulation entity data based on the various RISE simulation log results;
e. to provide a visualization tool for the RISE simulation log results; and
f. corroborate the use of the WebLVC protocol as a viable means of distributed simulation interoperability.

4.1 Design Methodology

In order to provide a mash-up of services and expose RISE into a traditional distributed simulation environment, an intermediary cross-platform approach was taken. As indicated earlier, different design approaches were considered at the commencement of this dissertation. One is to develop a generic wrapper that would allow multiple DEVS models to be published to the distributed simulation environment using DEVS/HLA. The second

approach was to create a thick client interface using C++ or Java that would accept RISE data and convert it into an acceptable form for distributed simulations. The third approach was to use the HLA 1516e WS and hosting the HLA 1516e WS on a server. The final approach was to use thin client distributed architecture of Figure 4 to expose the capabilities of RISE as a RESTful resource.

All approaches save one required changing the original model code of all the DEVS models. As this approach would require extensive man-hours of labour and the development of a testing interface to validate and verify the correct simulation execution of already developed models. Hosting the HLA 1516e WS, which is SOAP based, counters the development and application of RISE. The remaining approach was to create a thin client that harnessed the statelessness of RISE and convert the simulation results into a format acceptable for distributed simulation environment. This approach appears the best method as it does not obfuscate the original code, requires little or no modification to RISE, little or no modification to the simulation model code, little or no modification to the distributed simulation environment, and provides a modular and external interface.

Presently, MÄk has developed a prototype thin client prototype that allows single entity generation for a distributed simulation. As illustrated in Figure 5, JavaScript simulations can be interfaced to a WebLVC server to publish and subscribe update and reflect messages for a distributed simulation environment. Based on this premise, a thin client with full-duplex communication ability as shown in Figure 17 can be developed to provide a modular and external interface for a RISE and for a distributed simulation and is the same as Figure 6 but shows the reflect and update bi-directional data flow and another WebLVC server connecting a non-distributed web-based simulation. As the approach taken for the applications was originally coded in JavaScript, made use of HTML5 and utilized the built-in JavaScript engine of web browsers, the design approach was to continue the language selection and method of implementation.

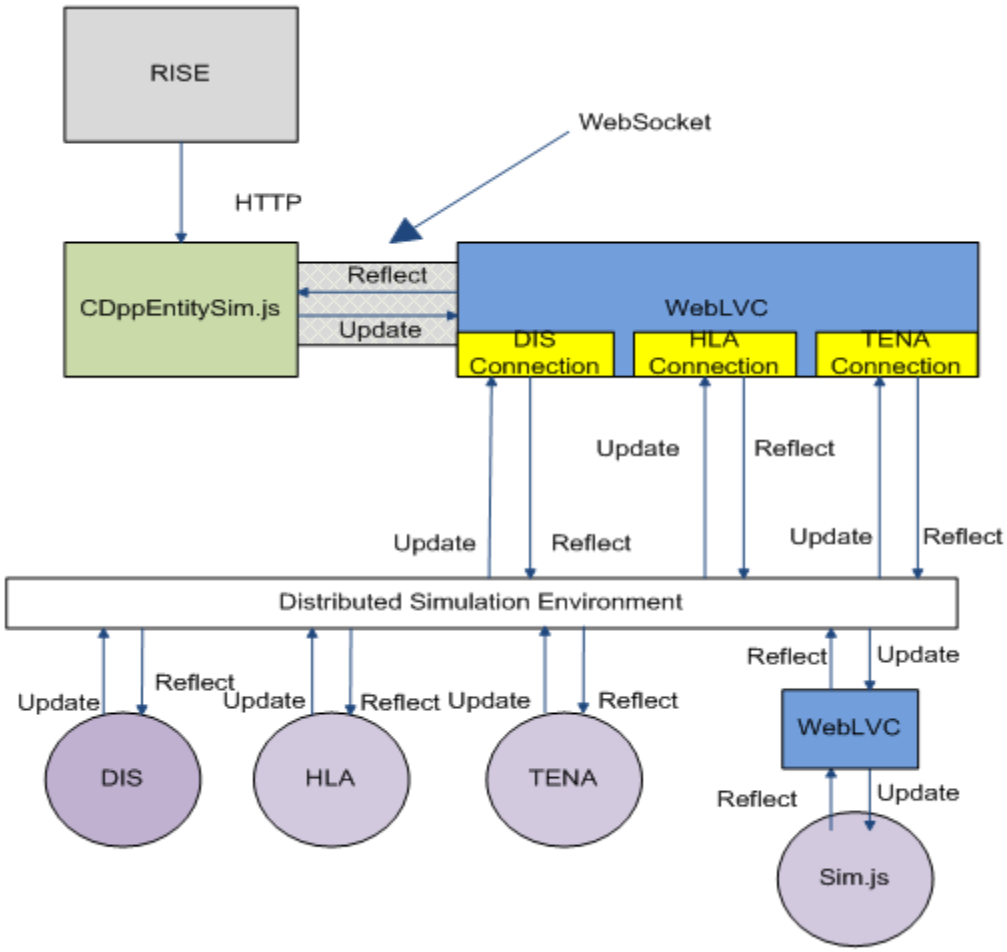

Figure 17. Modular and External Interface Diagram.

With regard to a global data interoperability and the relationship to other web based simulations and monolithic environments, this was illustrated in Figure 6. As stated earlier, exposing RISE to a distributed simulation environment would allow the introduction of highly complex world events for force generation and training. The technical challenges that are a concern in the development of any technique are as follows:

a. security;
b. simulation infrastructure;
c. networking;
d. stability; and
e. performance.

In particular, the developmental issues of a discrete event simulation had the following issues:

a. rendering complex visualizations to circumvent performance bottlenecks and threading issues;
b. atomicity in data log operations to ensure consistency; and
c. atomicity of data selection.

4.2 Implementation Details

The prototype entity simulation involved obtrusive HTML script combined with JavaScript as the main HTML webpage. Loaded as separate scripts were the entitySimulation.js file and the vehicleSim.js file which simulated the class structure for the creation of a single distributed simulation entity. Additionally loaded at runtime was an imageMap.js file which contained a function to return an image such as a F18 Hornet. Examination of the entity simulation was undertaken in order to discover how the entity was generated and to see what design patterns were made to accommodate the interfacing of distributed simulation to non-distributed simulation applications. Multi-threading capabilities were examined in order to evaluate the performance of a client application's main UI thread.

RISE is designed as a REST implementation where the server remains stateless and the client application such as a browser interacts and gains access to the server's resources through URIs. The results URI of each simulation execution is available through interaction with the browser, however, the CORS policy would prevent JavaScript access to the file without administrator permissions or having the server present an Access-Control-Allow-Credentials specification in its response header. Access to the RISE information for entity generation was crucial so an embedded method to access the information would have to be created.

In designing a web enabled interface, the problem can be decomposed into four major sections as follows:

a. discrete event simulation:
    1. access to the RISE results resource URIs,

2. decompressing and parsing of the different simulation result files into useable data,
3. storage of the data with acceptable memory requirements,
4. manipulation of the data into a format acceptable to be transmitted as the WebLVC protocol,
5. manipulation of the data into a format acceptable for viewing; and
6. manipulation of the discrete event time periods for time management coordination;

b. MÄk prototyped thin client application:
    1. modification of the entity simulation for multiple entities,
    2. control of the multiple entity generation,
    3. refactoring for unobtrusive HTML and JavaScript,
    4. redesign of the main HTML webpage for a discrete event simulation, and
    5. error correction;
c. visualization:
    1. provide local entity graphics representative of the discrete events simulation execution results file, and
    2. provide distributed simulation graphics representative of the discrete events simulation execution results file;
d. multi-threading;
    1. provide a multi-threaded technique;
    2. utilization of the multi-threading to prevent slow down of the main UI thread, and
    3. library creation for common methods accessible to all the threads.

4.2.1 Discrete Event Simulation Platform

4.2.1.1 RISE Simulation Results Access

RISE provides a URI resource results file that is generated as a compressed file of the simulation execution directory upon the successful completion of a CD++ simulation. In order to provide a client application capable of accepting the different results files, access to which user selection of the RISE results resource URI could not be known statically, but

only during run time. In other words, the user would manually traverse the URI to the desired results file. Having separate browsers or even tabbed web pages in the same browser could present usability problems as the separate pages could be closed and the user may load the wrong RISE resource results file. To correct this problem would be to embed the RISE URI onto the client side and have a default entry point to the main RISE resource URI.

With the administrator's right to select the file for storage or to open it through the operating system, selection of the saved file would have to be developed. Again, using the command line or explorer to access the file presents a problem on passing both the file information and internal data to the client application. The solution is to embed a file object handler which would supply read access to the file and the properties of the file.

4.2.1.2 Decompressing the Simulation Results File

Depending on the platform used by the user, a third party tool could be used to uncompress the file. This would require the user to be knowledgeable about which file is to be extracted and require the user to store the uncompressed file into the file system. However, this defeats the presentation of the client side as the sole mash-up of service between the different WS services. In order to overcome this dilemma, a scripted and embedded uncompressing tool is proposed to be utilized. This allows the client side to retain the data without having to save, uncompress and re-open the required files.

Because RISE stores the file in a compressed structure, a means would be required to inflate the file and determine the internal files. Again, this usually is resolved through user selection as with the many different resource result files accessible on RISE, the selection method to access the CD++ simulation execution remains textually documented on the RISE URI pages. In the state selection of the pages, a user can select either a DCD++ service or a Lopez service. Thus, we propose presenting a user selection technique on the client side for the identification of these user selected services which generates the ability to correctly extract the data.

The inflated data log file whose format extraction is predetermined by the user is a text file consisting of lines of data separated by varying lengths of white spaces. In the Lopez format each line is an output message, while the DCD++ format is a combination of the input, output and internal messages. Lines within each format can also be duplicated as the CD++ simulation engine stores the results of the execution in a line by line format in memory and directly writes this memory space to a file upon completion. In order to prevent repetition of entity data, a proposed means has to be developed to eliminate duplicates from the data prior to separating the data in its respective event data containers. Upon completion of the duplicate removal, the results log file data would be parsed and separated by new lines and white spaces and stored into individual containers indexed with the line position in the result's log file.

4.2.1.3 Unique RISE Simulation Results Entities

Concerning distributed simulation, the WebLVC server requires a unique object name, entity type, entity identifier, position, and vehicle information parameters in order to track the entities and publish the entities to other distributed simulations. As Cell-DEVS could have an infinite three dimensional grid reference, the choice arises to present the Cell-DEVS grid as a single entity or as multiple entities. Presenting as one entity would not present the dynamics of the Cell-DEVS simulation to other enabled distributed simulations so the approach would be to present the Cell-DEVS grid as separate entity for each grid cell coordinate, where the a cell having value becomes an entity. This requires that each entity have a positional reference from the Cell-DEVS grid cell origin (0, 0, 0) in latitude and longitude. Following on this implementation detail, a bearing, distance and speed of advance is proposed to be calculated based on the data. This is illustrated in Figure 18 which shows the three distinct event periods where Cell-DEVS grid cells are populated with some value as indicated by the dot. Positionally, each cell has a relative bearing and distance defined by the spacing distance of the grid cell from the origin. This positioning of the Cell-DEVS neighbourhood remains in a heads-up view.

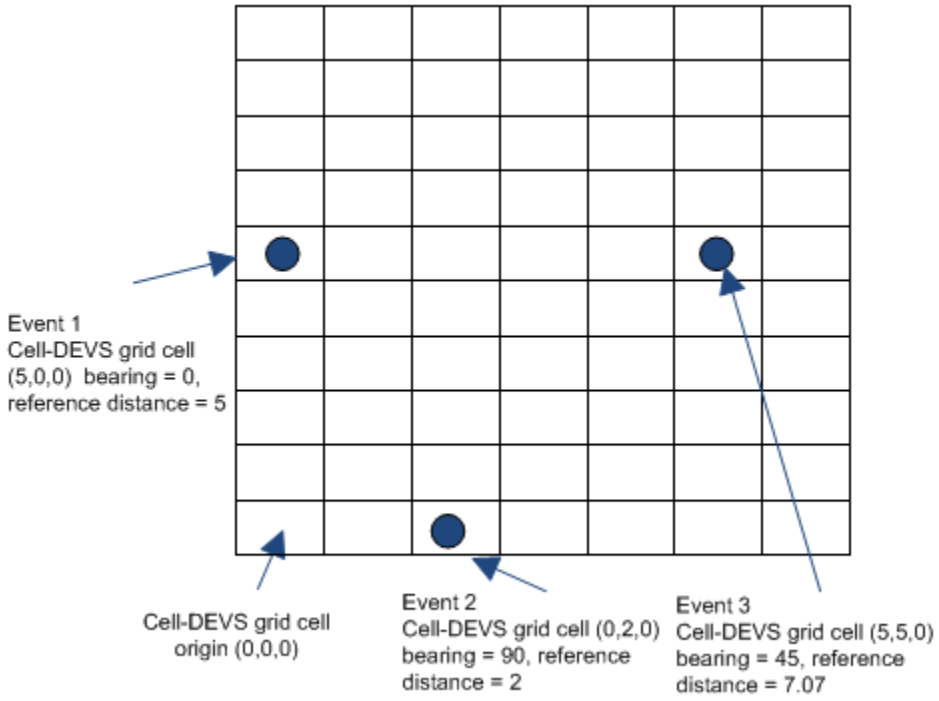

Figure 18. Cell-DEVS Cell Neighbourhood Positional Bearing and Distance.

4.2.1.4 Controlling The RISE Simulation Results Event Periods

With multiple entities and multiple calculated latitudes, longitudes, distances, and bearings, a means must be established to present this data to the user. Tabling the data on the MÄk prototyped left pane tableau would indicate only the latest entity, however, as this assumption is based on only having a single entity being transmitted. Thus, the left pane tableau will display the latest entity data while a Dijit embedded tabbed pane is proposed to provide all the entities and their informational data. Therefore, there will be three main panes – the Map pane, the RISE Server Webpage pane and the Processed Data pane.

The data obtained from the RISE results resource URI are event driven where at certain times, events occur. This dissociation of the client side simulation time to RISE event time would require consideration as the event rules and the span of the event time periods are entered by the RESTful-CD++ simulation developer. The span of the events therefore could be generated for faster RESTful-CD++ simulation execution rather than real-time execution. For example, the RISE simulation execution Lopez FallRock model, has a time span of event

execution of 1.2 seconds while the DCD++ fire model has a time span of event execution of 7 hours 23 minutes and 11.248 seconds. Controlling the event time of the simulation would allow debugging, correct the default simulation time event parameter, and activate the RISE data being populated. The RESTful-CD++ event time would be disassociated and have no relative effect on the client side's distributed simulation time. In order to overcome this predicament, we propose having the RESTful-CD++ event time manipulated by a scale factor to speed up or slow down the processed RESTful-CD++ data upon selection by the user to populate the RESTful-CD++ data into the distributed simulation.

Again the RESTful-CD++ simulation developer was free to determine the value of each grid cell space and the meaning of each value. These values are based on the rules that allow the execution of the Cell-DEVS simulation and are contained within the MA extension file. As this file is not accessible to the general user but only to the RESTful-CD++ simulation developer, we propose allowing a default empty grid cell value to allow specification of the Cell-DEVS output value and therefore, the creation of the WebLVC delete entity message. Similarly, a RESTful-CD++ simulation developer can specify the output name of the coupled Cell-DEVS model. To overcome this predicament, we propose specifying a default output port that can be modified at run-time prior to the decompression of the RESTful-CD++ resource results file.

4.2.2 Thin Client

4.2.2.1 Dojo and Dijit Ajax Operations

The entity simulation developed by MÄk is a combination of HTML, JavaScript and Dojo for AJAX calling. It was designed and written in JavaScript, a dynamic object-oriented general purpose programming language, and contained both Dojo and Dijit components. Dojo is a rich library containing the core and most non-visual modules to allow abstract event handling through its simple API calls. Dijit, the UI library for Dojo, allows the reuse of preprogrammed widgets to provide key functionality by having a tremendous wealth of high quality and feature-rich form elements. As the original entity simulation was a

prototype and may be continually developed, any errors that were present would be corrected prior to developing the client side as a mash-up of services.

The display for the entity simulation is proposed to continue to use the Dojo and Dijit libraries to provide event handling of the map selection and loading, the unloading of the client side, widgets for display on top of the map and mouse event handling. Additionally, the original left pane tableau would have to be refactored for the acceptance and display of RISE specific data and information labels for the user. Such labels would be the compatibility of the user's browser to enable WebSockets and web workers and the properties and time span of the file obtained from RISE along with the aforementioned changes.

4.2.2.2 Unobtrusive HTML and JavaScript

The file structure of the original MÄk entity simulation is proposed to be restructured for unobtrusive HTML and JavaScript so that there is as large as possible separation of content between the languages. The window and document function calls would be housed in a separate file from the HTML and common non-window / non-document functions would be housed in another separate file for use with the web workers as discussed below.
An application object, housed in its own file, is proposed to provide an abstraction for the entities' requirement for a unique name. Additionally, the WebSocket interface would be made accessible to all the entities that would be created instead of a WebSocket interface per entity. Realistically, if the RISE results resource file contained a large multi-dimension grid infrastructure, having a WebSocket per cell coordinate would quickly over complicated the client application and possibly saturates the limits of the WebLVC server, as it expects one WebSocket per client application.

As the main common interface to the WebLVC, MÄk vrlink.js file is presently only DIS compatible for reflected entities and is coded for the DIS vehicle entity. Thus, the client application would retain the vehicle class object as checking is performed on the outgoing

JSON packet in order to ensure the data is correct and the incoming data is marshalled into a DIS vehicle entity.

4.2.3 Visualization

The ESRI ARCGIS map would continue to be utilized for the entity generation and positioning. Enhancing the display of the entity would be a local grid cell signature layer of the entities. As the RESTful-CD++ simulation engine has dealt with models such as fire, flood, and smoke dispersion, having the populated grid cell coordinates populated with an event visualization provides a force commander with a realistic and dynamically changing image of the event rather than a static graphic. In this regard, an open-source library called heatmap.js is proposed to be used as the event simulation medium.

As the actual entity details are defined by the entity type in a numeric enumeration array, any possible combination of entities could be created above and beyond the supplied images of different vehicle type. With this in mind, the static image of a vehicle type such as the supplied American F18 will be altered to a simple graphic symbol to return as the RESTful-CD++ entity image. The visualization.js file will house these graphic creation functions allowing for modularization of the code.

It must be noted that other distributed simulation client applications will display the entities as whatever graphics or images that those developers choose. Presently, the 2D, 3D and OpenLayer applications supplied by MÄk, these client applications present the entities as coloured dots. The MÄk prototype entity simulation is coded to define an environment entity as a Ford Contour sedan based on the array positioning of the entity type.

4.2.4 Multi-Threading

As JavaScript is single threaded in an event-loop, heavy processing may cause the main UI thread to become over-tasked and result in a “script-not-responding” modal dialog. In order to overcome this non-responsive warning, we propose to have web worker threads committed to the extraction, storage and processing of the data.

The benefit of using web workers can be demonstrated using Oliver's RayTracing application, available on the internet, [Ray01], as the application shows the benefit of multi-threading. It also demonstrates that the boundaries as the communication requirement can off-set the value of limitless multi-threading.

Table 2 tabulates the multi-threaded web workers in action to produce the RayTracing application output with the confidence intervals calculated [Ban10]. Five trials were set up to evaluate the mean processing time required for the ray tracing application with increasing numbers of web worker threads. Figure 19 illustrates this tabulated data, and allows one to see that for this particular application, the benefit of multi-threading, as designed for the RayTracing application, is limited to four threads. Any further addition of threads did not demonstrate a significant decrease in the rendering time. It can also be inferred that though an increase in threads may continue to process increases in efficiency, the latency of the data passing through the message channels becomes the limiting factor.

Because of the security aspects of the threads, data can be passed either as reference or as value. In JavaScript, the passed by reference method will pass data from one memory space to another; however, the losing memory space no longer has access to the data. The data is lost to the assigned variables though the communication speed of passing the data dramatically increases. Since the web workers have control of the arrays and objects for data manipulation rather than the main UI thread, we propose that the web workers retain control of the full arrays and objects and that if required, only local scope copies of the pertinent sections of data be passed by reference otherwise all data will be passed by value. This proposed implementation is due to the fact that unlike a C code style pointer which allows multiple references to a memory space, JavaScript passing by reference only allows one reference to a memory space at any one time.

Threads	Time (s)					Average (s)	Standard Deviation	Confidence Interval (95%)
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5			
0	1.765	1.769	1.778	1.758	1.8	1.774	0.016	0.014
1	0.786	0.754	0.704	0.77	0.744	0.7516	0.031	0.027
2	0.398	0.376	0.371	0.387	0.397	0.3858	0.012	0.011
3	0.358	0.381	0.318	0.329	0.316	0.3404	0.028	0.025
4	0.321	0.256	0.287	0.233	0.261	0.2716	0.034	0.029
5	0.285	0.266	0.275	0.257	0.253	0.2672	0.013	0.011
6	0.287	0.259	0.24	0.265	0.23	0.2562	0.022	0.020
7	0.287	0.225	0.271	0.232	0.275	0.258	0.028	0.024
8	0.293	0.251	0.249	0.22	0.259	0.2544	0.026	0.023
9	0.297	0.225	0.247	0.272	0.246	0.2574	0.028	0.024
10	0.317	0.311	0.238	0.228	0.236	0.266	0.044	0.039
11	0.33	0.259	0.226	0.244	0.225	0.2568	0.043	0.038
12	0.31	0.216	0.237	0.255	0.27	0.2576	0.036	0.031
13	0.302	0.235	0.229	0.257	0.264	0.2574	0.029	0.025
14	0.317	0.246	0.249	0.229	0.265	0.2612	0.034	0.030
15	0.3	0.233	0.246	0.223	0.269	0.2542	0.031	0.027
16	0.295	0.243	0.236	0.272	0.26	0.2612	0.024	0.021

Table 2. Ray Tracing with Web Worker Threads.

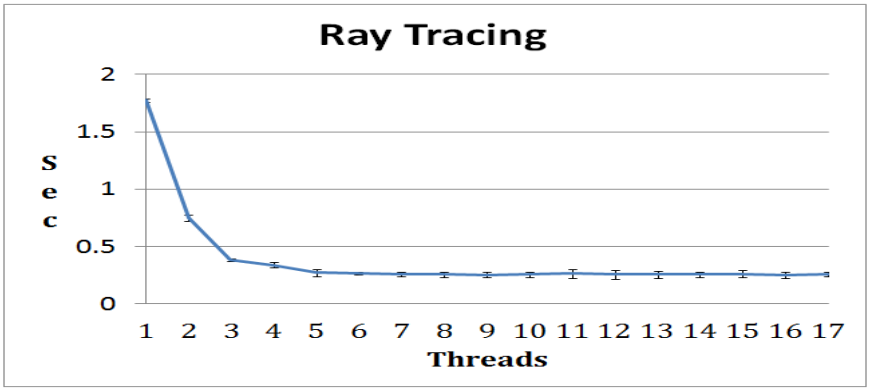

Figure 19. Ray Tracing Web Worker Thread Rendering Time.

4.3 System Architecture

The system architecture can be decomposed into a HTML webpage, nine JavaScript files and MÄk's prototype software that transmit and receive data from the WebLVC server.

4.3.1 JavaScript Event Handlers

In executing the HTML webpage, JavaScript code that is not in functions and is placed in the head section of the HTML page will run before the web page content is loaded. Placing the same JavaScript code in the body of the web page, will execute as the page is loading. Placing the same JavaScript code at the bottom of the web page, will execute after the HTML file has loaded but possibly not before images are loaded. In all cases, the entire web page may not have been loaded when the JavaScript is executed. In order to ensure that the JavaScript executes after the web page has fully loaded and to guarantee that images such as a map are loaded, we propose attaching event handlers to the head tag itself upon the map's completion of its loading event ensuring that variables are instantiated at the correct time. The event handling connection to the function handlers is depicted in Figure 20.

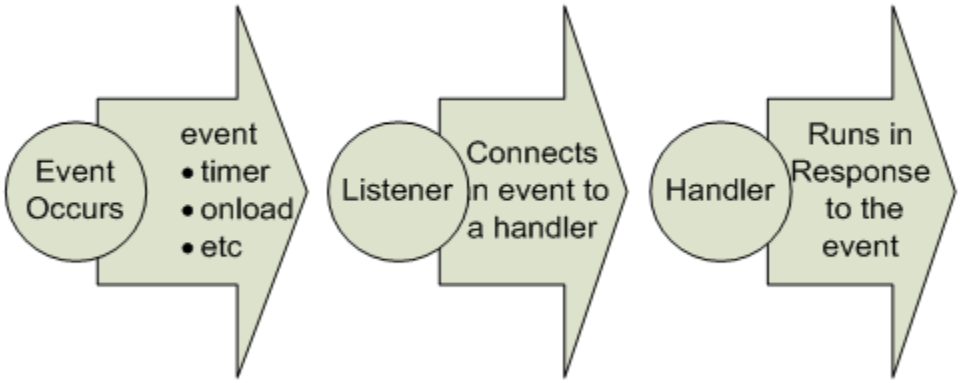

Figure 20. Event Handling Process.

The onload event handler will only execute the JavaScript upon the successful completion of downloading all the images, scripts, and style sheets into the browser window. Similarly, an event handler would also be triggered upon the user refreshing or moving away from the web page. In this case, we propose using the Dojo event handler for the onload and unload events. Similar to the on load and unload events, event handlers are proposed to be

attached for the mouse movements and its interactions with the map and with the onresize event of the map. With JavaScript as an event-driven language and sequentially run, the execution of the CDppEntity Simulation as a thin client must follow general process rules of HTML and can be best expressed through a page loading sequence diagram as in Figure 21.

As per the sequence diagram, the CDppEntity Simulation first loads the HTML webpage and the parsing of the HTML document starts. The imported scripts and the Cascading Style Sheets (CSS) are downloaded and parsed. The next event is the internal CSS and internal JavaScript is parsed and run. When the parsing is completed and the document is ready and loaded by having the images are fully downloaded, the Dojo events onload is fired setting the initialization routines. Because of the requirement for WebSockets, the initializing function is designed to check for browser compatibility with and the handlers for the DOM objects are set. Also the WebSocket and random number are initialized as required by the distributed simulation. Once the map graphics are downloaded, the storage container for the reflected entities from the distributed simulation is initialized. Map interactions by the user are left for another Dojo event handler.

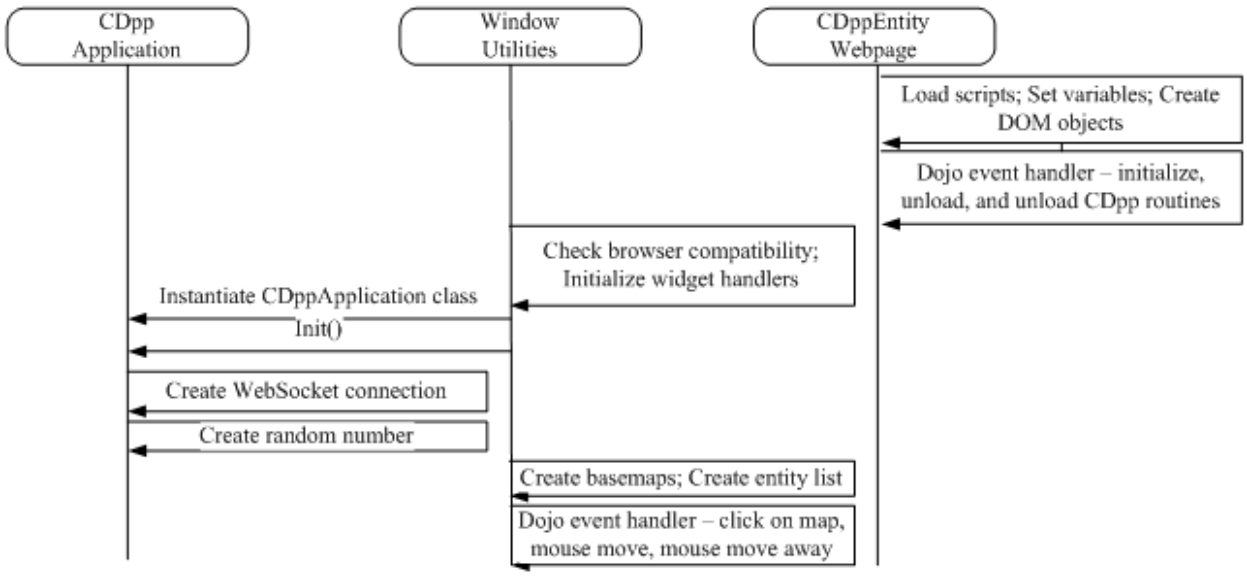

Figure 21. CDppEntity Simulation Page Loading Events.

4.3.2 Multi-threading System Approach

The proposed system architecture defines the separation of JavaScript from HTML for unobtrusive code. Because of the processing of the RISE resource results file, multi-threading is proposed to allow the user full interaction with the main UI thread, while background processing is accomplished. Thus, the client side will house three threads - the main UI thread, a RISE web worker thread, and an Entity web worker thread - to prevent high single CPU utilization. The main UI thread, because of thread security, contains the classes and functions that interact or indirectly interact with the window object - the web page and the DOM API. It must be noted that the main UI thread cannot be guaranteed to be selectively responsive as the web workers cannot block the main UI thread and make it totally non-responsive, but they still consume CPU cycles and can, therefore, make the total system less responsive under high processing.

Common to both the UI thread and the web worker threads is an utilities.js file which houses the common classes, message objects, non-window object functions and manipulation functions. The two web worker thread files are the riseWebWorker.js file and the entityWebWorker.js file respectively. The classes that comprise the main UI thread are proposed as follows:

a. CDppEntity Simulation HTML web page;
b. cdppApplication.js;
c. cdppEntitySim.js;
d. vehicleSim.js;
e. visualization.js;
f. restfulResource.js; and
g. windowUtilities.js.

The instantiation of the web workers thread is called from the main UI thread and communicate through message channels to and from the parent thread to the child threads. Upon a message transmission, the event handlers of the threads are fired and functional calls are then processed depending on the message value. This is illustrated in Figure 22.

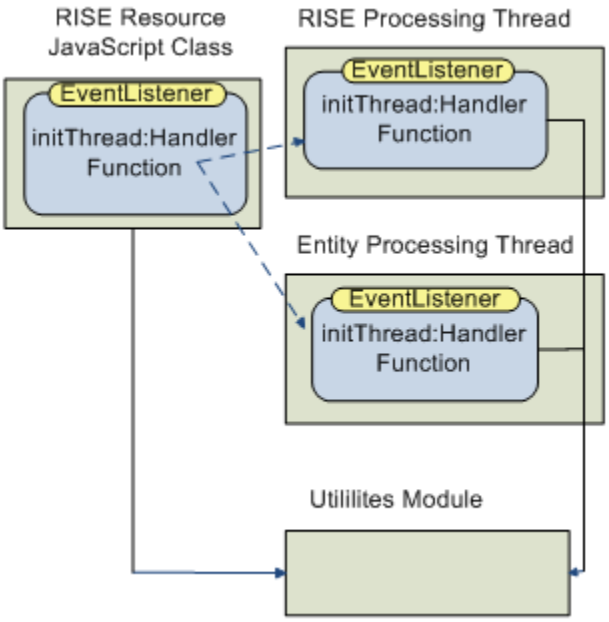

Figure 22. Thread Event Handling and Processing.

4.3.2.1 Multi-threading Data Processing

Coordination is required when transferring data between threads. The design of the coordination in particular is important in order to ensure that data is present and valid prior to processing. Thus, a sequence diagram can communicate how objects interact with each other and can be used as a "requirements document" to showcase the prerequisites for a future system implementation.

In Figure 23, the user's interaction with the webpage describes the proposed events for the processing of an epoch selection execution of the RISE resource results file while Figure 24 shows the sequence for non-epoch data execution. The figures are designed to show how the objects and calls interact with each other in each particular use case scenario elaborated below in section 4.3.2.2 Multi-threading Data Processing Concerns. Function calls to the utilities library are not displayed, but the functions do contain module library calls and should be realized as important. Upon selection of the browse command button to

access the compressed RESTful-CD++ results file downloaded from RISE, a RISE and Entity web worker threads are created and populated with parameters from the parent UI thread. Upon its completion, the graphic layers would be loaded on top of the map graphic layer to house the graphic symbols or images. The compressed file is decompressed and passed to a utilities library module function which extracts of the data. Properties of the files accessed are then written to the output label of the web page for user information.

Event handlers of each thread, depending on the required function, would process the message command, and return the desired data set by the parameters of the call or further process the data and return the processed data to the calling thread. Upon successful reception of the data by the calling thread, its event handlers would set member variables or call functions such as creating entities or displaying the entity data information in the grid view pane. Calls to threads or functions are indicated by the solid lines while return replies are indicated by a dotted line. Boxed solid lines refer to one or more function calls that are required within the thread to process the data received and are actioned by the event handler's reception of a message.

The windowUtilities.js file is proposed to contain functions and calls that allow the determination of the log file from the RESTful-CD++ results file based on the user's selection. The inflating of the file is proposed to be accomplished by an open source uncompressing library called bitjs [Bit13], which upon successful inflation, stores the data of the expected log file into an ArrayBuffer. This typed array would then be converted into a unit8Array format in order to access and manipulate each of the RESTful-CD++ log file lines of data. The array would then be posted to the Rise web worker thread for data extraction along with the user's selected latitude and longitude and the scale factor value. After parsing the data, the web worker would respond with the initial data values required to start the simulation.

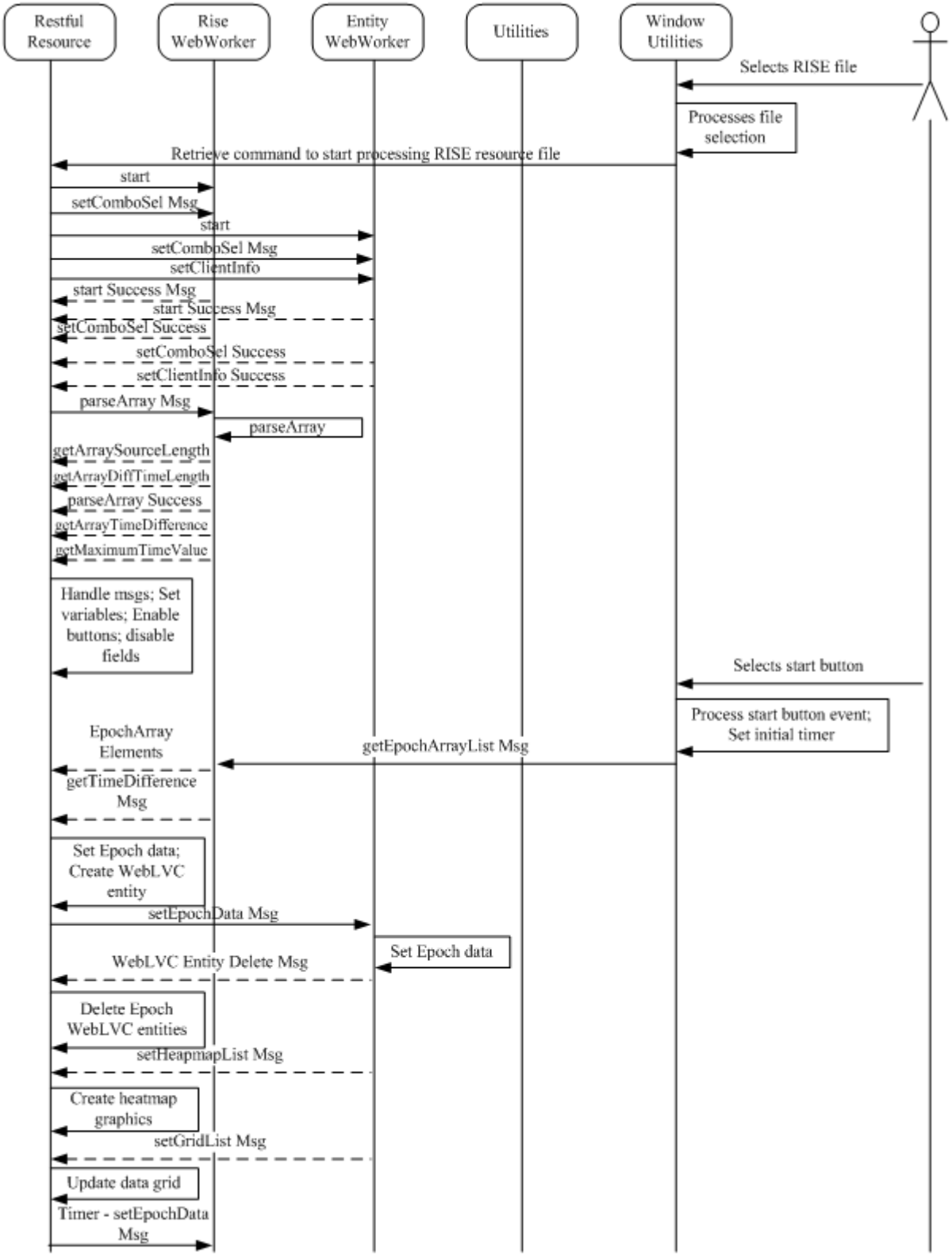

Figure 23. User Interaction for Epoch Event Processing.

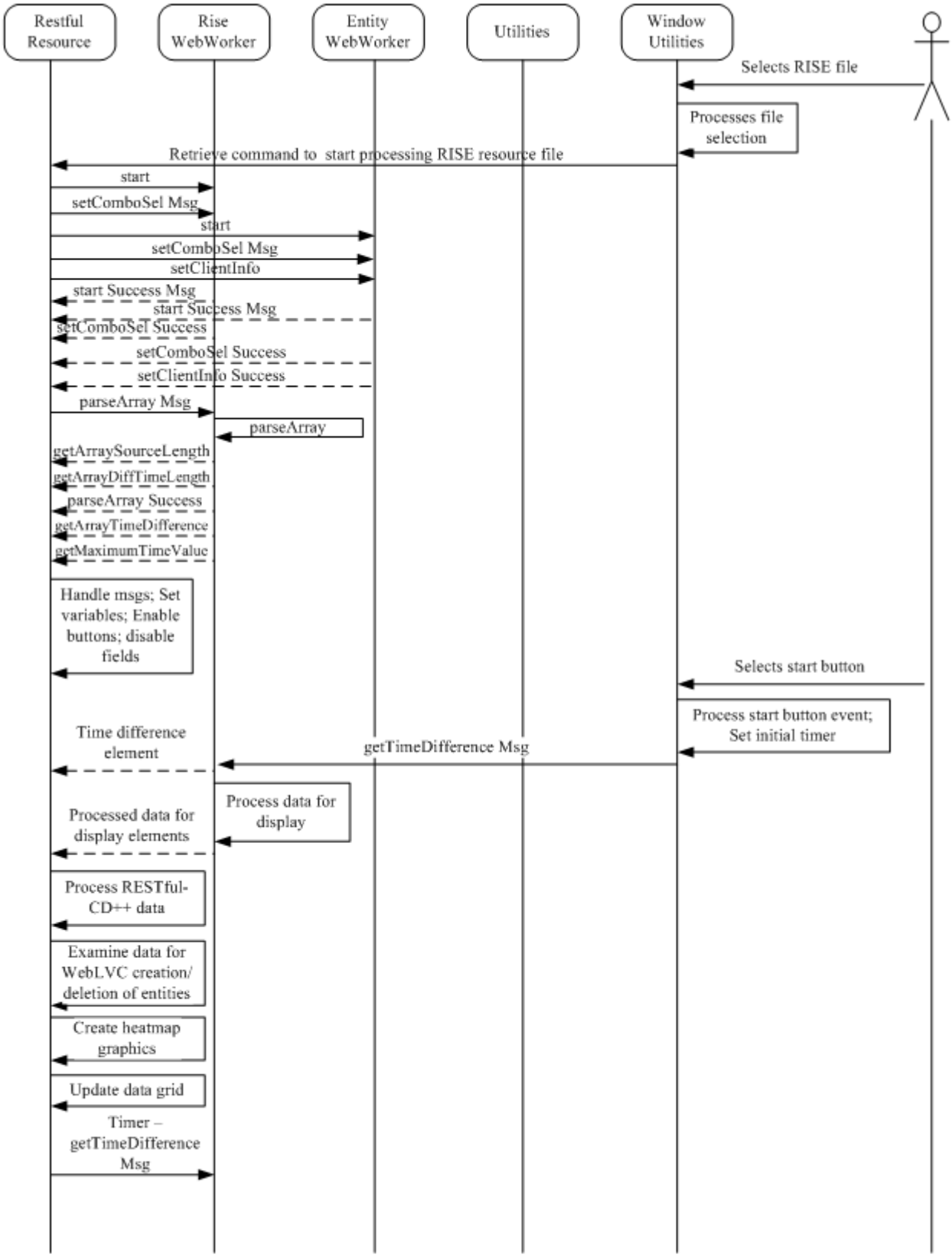

Figure 24. User Interaction for Non- Epoch Event Processing.

4.3.2.2 Multi-threading Data Processing Concerns

Based on the types of already generated RESTful-CD++ log files, the event data can be described in possible extremes. The rapidity of the data events such that the simulation is completed before the user realizes the simulation has started or the converse extreme where the events occur over long durations. Over-coming this paradigm, the user would have the ability to select a RESTful-CD++ time event scale factor that would allow the increasing or decreasing the event period.

Another possible implementation difficulty would be the data event period or the amount of data per epoch event period. The events happening to each of the RESTful-CD++ grid cell coordinates could produce a small epoch time period with multiple events or in corollary, a large time period with a few events interaction. Based on this theory, two different selections of epoch processing would be available for the user's selection depending on the desired results. The user would be able to select all the fully preprocessed data per epoch or the RESTful-CD++ converted data where the UI thread processes the data for visualization and interfacing with the WebLVC server. Each method would serve different purposes to minimize communication problems such as data not being available at the correct period; epoch data selection would be for a large amount of entities, while non-epoch data selection would be for large amount of event periods.

4.3.2.1 Event Data Dynamics

Upon the start command, a timer function is proposed to be created and at each successive interaction of the timer, clocked on a combination of the event time periods and a user selected RISE time event scale factor, the event data would be passed from the RISE web worker to the RestfulResource class for population of the graphics on the web page. In epoch processing, for example, each event data would be converted in the web workers from the original log format into real world positional information for visual display and transmission to the WebLVC server. Upon the stop command, all WebLVC entities would be

deleted from the distributed simulation and local graphics erased. The web worker threads would terminate and memory space would be collected by the garbage collector.

The client side is proposed to keep track of the entities already created by collecting entity object name, entity identifiers, entity type and their timers for created for the RESTful-CD++ grid cell coordinates. The client side is proposed to create a WebLVC entity, for each RESTful-CD++ grid cell coordinate based on the value of the RESTful-CD++ grid cell coordinate for the output Y messages of the RISE resource results file. However, there may be more than one output port with a positive value and it is proposed that if a string matches the user's entered value an entity is created. The default value for this item is proposed as an array of characters representing the possible output values and is defaulted to zero but can be changed as each RESTful-CD++ developer can arbitrarily specify the output values for the RESTful-CD++ simulation execution. Similarly, the client side is proposed to delete a WebLVC entity, if it exists, for each RESTful-CD++ grid cell coordinate if the value of the RESTful-CD++ grid cell coordinate matches the user's entered value for the Delete Entity Value textbox. The default value for this item is proposed as a zero value but can be changed as each RESTful-CD++ developer can arbitrarily specify the input, transition and output values for the RESTful-CD++ simulation execution.

4.4 System Interface

As a mash-up of services, we propose that the CDppEntity Simulation have separate distinct interfaces to which it must connect. The first interface grants the access to the RISE server manipulation of the RISE resource results file. The second interface is script access to items such as the ESRI ARCGIS map data on which the graphics and entities are layered. The third interface is the access to the WebLVC server for the production or consumption of distributed entity simulation data.

4.4.1 RISE Iframe Interface

The RISE interface is proposed to be embedded as an iframe within the Dojo content pane. Since the RISE outputs its information as an html or xml document, the iframe is used, as

designed, to contain the html document in a nested browsing context. Being embedded, the source of the iframe can be easily gained from the user's input selection of the RISE server's location. With the now embedded RISE resource URI, the user is free to examine the different URIs and, if desired, retrieve a RISE resource results file to the local file system for data extraction and manipulation. Of course, the RISE resource results file is not mandatorily required to be downloaded again if the RISE resource results file is already stored within the local file system.

4.4.2 Internet Access

The ESRI ARCGIS maps are used as the main starting point for the client side in order to provide the initialization stage for the entity execution. Thus, failure of the map to load prevents the initialization of the simulation as the Dojo event handlers await the map's onload event to trigger the initialization routine. As the map is continually being downloaded as the user changes the scale of the map, a constant internet connection is recommended.

4.4.3 WebLVC Communication

Lastly, the client side would normally begin its interface to the MÄk WebLVC server through TCP/IP connection. The software for creating a web-based distributed simulation is MÄk's vrlink.js library module. This file, called by a client application, houses the classes and interfaces for the creation of a WebLVC WebSocket, its serialization / deserialization functions and defines the attributes and methods that allow a client application to access the services of the WebLVC server. As the WebLVC server is a modification of the MÄk VR-Exchange product, vrlink.js is presently prototyped only as a DIS distributed simulation interface.

In order to produce or consume entity information from the WebLVC server, the data must be transmitted and received as a JSON-based object. Using MÄk's Message Inspector application, the JSON object can be examined as proposed in Figure 25 which shows a JSON-based entity message from the RESTful-CD++ grid cell coordinate (9,2,0) for a DIS

environment in accordance with the required DIS enumeration [DIS11]. The information presently expected by the WebLVC server is as a vehicle entity with attributes such as velocity, acceleration and damage state, which in this case, would be defaulted to a zero value. The type of JSON message interchanged with the WebLVC server can be any, other, attribute update, interaction, connect, or object deletion. Differentiating between an attribute update and interaction message, for example, an interaction message would inform the distributed simulation applications that a F18 is circling with a radius of 1000 around a specified world positional coordinate while the attribute update is the default type of message for instantiating and updating an entity. The draft specification of the WebLVC protocol, version 0.1, only documents the attribute update.


```
{
"AccelerationVector": "[0,0,0]",
"AngularVelocity": "[0,0,0]",
"DamageState": 0,
"DeadReckoningAlgorithm": 4,
"EntityIdentifier": "[1,1,920]",
"EntityType": "[4,1,0,0,0,0,0]",
"ForceIdentifier": 0,
"Marking": "WebLVC-CDpp",
"MessageKind": "AttributeUpdate (1)",
"ObjectName": "CDpp 69354_9_2_0",
"ObjectType": 1,
"Orientation": "[1.818082012506282,-0.7780143250425958,-
3.1398729077441]",
"Timestamp": 1358.53041601181,
"VelocityVector": "[0,0,0]",
"WorldLocation": "[1108268.601076215,-
4345117.870447309,4520465.02507074]"
}
```


Figure 25. WebLVC Protocol JSON Entity.

The entity once published, requires a constant update or heartbeat in order for the WebLVC server to realize that the data is not stale and that the sending application is still on-line. This entity update is performed through a timer function created by each entity. Deletion of the WebLVC entity is performed in combination by two methods. The first method is by cancelling the entity's update timer to indicate stale data, and the second method is by submitting an entity deletion message shown in Figure 26 which informs the

WebLVC server to remove the entities from the distributed simulation. As shown, the message requires the unique entity identifier, the type of message, and the entity's object name. Local entity graphics would be deleted upon a deletion message or removed by comparison with the WebLVC reflected entity list.


```
{
 "EntityIdentifier": "[1,1,28270]",
 "MessageKind": "ObjectDeletion (4)",
 "ObjectName": "CDpp1_42942_28_27_0"
}
```


Figure 26. WebLVC Protocol JSON Deletion Message.

Chapter 5: CDppEntity Simulation

Using a combination of the new HTML5 technology, RESTful-CD++, and prototype WebVLC clients, we have developed a client side interface between the RESTful-CD++ service and a WebVLC server, allowing the high fidelity visualization of cellular automata data and providing a new distributed simulation interface.

5.1 Developmental Environment

The following sections describe the hardware and software development environment used to create the web-based CDppEntity Simulation.

5.1.1 Hardware Platform

The software developed for the RISE HLA Interface was developed on a custom built Intel platform. The computer housed a Z68 motherboard containing 16 Gigs of memory, a K7-2600 CPU, a Radeon High Definition (HD) 6850 video card and a gigabit Ethernet connection. The network was connected to a high-speed cable modem via CAT6 Ethernet cable. The WebLVC server was hosted on a server containing 2 Gigs of memory, a Pentium 4, and a gigabit Ethernet connection.

5.1.2 Software Platform

The software was developed on a Windows 7 64 bit operating system using an Eclipse Enterprise Edition (EE) 64 bit software development Integrated Development Environment (IDE). Client browsers hosted were Firefox 19.0, Chrome Version 25.0.1364.97 m and Internet Explorer 10. The WebLVC server operated on a Windows New Technology (NT) 2003 server.

5.1.3 Integrated Development Environment

The development environment used to create the CDppEntity Simulation was Eclipse Java EE IDE for Web Developers. The version was Juno Service Release I. Jetty Web Tools

Platform (WTP) Adaptor, version 1.0.0.20120301137, was used to provide local WS for deployment. WebLVC Server 1.0 was used as the client interface for the distributed simulation entities.

Execution and debugging of the program was conducted through the browser developmental platforms such as the Firefox application with the FireBug plug-in. This allowed the flow of execution to be analyzed and each line to be step traced in its process execution for code coverage. It must be noted that JavaScript is a high-level, dynamic, untyped language that is interpreted at runtime by the client browser and is not statically compiled. As such, programming and syntax errors are not statically caught as would happen with a compiler and which would aided in the determination of compilation errors, allowing the developer is easily recognize both syntax and scope errors. Conversely, because of the runtime execution of the interpreter, inherent errors from method calls were returned and execution ceased without warning. This required extensive man-hour labour to step through each line of code, identify the exact error, correct it and retest. Simple errors such as syntax or misspelling of a variable name hampered and defeated proper code execution, for example.

5.1.4 Browser Testing

As only Chrome implements the allocation of both temporary and persistent memory storage, it was decided for cross browser capability to use the temporary stack space to store data. Though the idea of BLOB and the window.requestFileSystem function may be the future of browsers for file or quota management, heap or stack space is still the standard operating method. In order to create a mash-up of services between the RISE and the WebLVC server, the browser was checked in order to identity the limits to which an array could be populated with the data of the RESTful-CD++ output log file. In order to test this capability, a function was developed to test the limits of the number of elements that an array could hold prior to generating a “out of stack space” error. In the code snippet of Figure 27, an array memory allocation test was performed. Using Chrome, Firefox, and Internet Explorer 10, the results were approximately 55,000,000 elements, approximately

95,000,000 elements and exactly 77,972,449 elements respectively. It must be noted that Chrome “crashed” on any significantly larger number, while Firefox, though it did not ”crash”, it failed in the execution of the code and returned from the method without warning, prompt or any further continuing execution. Internet Explorer was the one browser to correctly interpret the try catch statement and threw an alert with the specific number of elements at the “out of stack space” error. Whether the element was a large string, small string or number, the number of elements was consistent for each browser but this number was not consistent across all browsers. The function was then modified into five arrays to see if the maximum allocation was per array or rather the memory limits. As suspected, the maximum memory was tested. This meant that for the client side, the CDppEntity Simulation, the elements of the populated arrays multiplied by the number of arrays for a RISE resource file must not exceed 55,000,000 elements. This does not include the map, entities, classes or any other installed components, but rather gives an idea of the maximum size of the RISE resource results file to which a browser can manipulate.

```
function getMemorySize()
{
	var elements = new Array();
	try
	{
		for (i=0; i<95000000; i++)
			elements[i] = "0 / L / I / 00:00:00:000 / fire(01) for
fire(0,29)(31)";

		alert("size of the array is: " + elements.length);
	}
	catch (err)
	{
		 alert(err.message + " " + elements.length);
	}
}
```

Figure 27. Browser Memory Allocation Test.

As the browser could be heavily loaded with content from the RESTful-CD++ resource file and its manipulations for visualization, a function was created similar to the inherent sizeof() method in the C language. The function is limited as it is a compromise and a slow

testing method as it can check an object's size only if it does not contain an internal private functions but just other objects. This can be seen in
Figure 28. Various large arrays and objects used in the program are documented below in Table 3 for the RESTful-CD++ DCD++ fire model and the Lopez occupancy model. Based on the data examined, one can infer that the memory limits of the browser are not met and that the client application will not over-stress the browser and cease its execution. In the worst case scenario where the garbage collector fails and all elements remain in memory, the containment of the data would at maximum be 0.137 GB compared to Chrome's limit of 2.76 GB as discussed above. The largest log file for the Cell-DEVS models presently on RISE is the Lopez Occupancy model.

```
function getObjectSize( object )
{
        var objectList = [];
        var memory = [ object ];
        var nBytes = 0;
        while ( memory.length )
        {
                var value = memory.pop();
                if( typeof value === 'boolean' )
                        nBytes += 4;
                else if( typeof value === 'number' )
                        nBytes += 8;
                else if( typeof value === 'string' )
                        nBytes += value.length * 2;
                else if( typeof value === 'object' && objectList.indexOf( value )
                === -1  )
                {
                        objectList.push( value );
                        for( index in value )
                        {
                                memory.push( value[ index ] );
                        }
                }
        }
        return nBytes;
}
```

Figure 28. Function to Return the Number of Bytes.

Location	Type	Name	Purpose	Size (bytes)
cdppEntitySim.js	class	CDppInitializer	Default WebLVC entity initializer	230
cdppEntitySim.js	class	CDppInitializer	After parseURL function	256
cdppEntitySim.js	class	CDppEntity	WebLVC entity	3268
restfulResource.js	string – Fire Model	unzippedText	Uncompressed data – Fire model	6,486,118
riseWebWorker.js	array – Fire Model	dataInput	unzippedText split into line of text	6,394,216
riseWebWorker.js	array – Fire Model	arr	Elimination of duplicate lines of text	5,066,610
riseWebWorker.js	array – Fire Model	dataArray	RESTful-CD++ parsed and manipulated data	217,350
riseWebWorker.js	array – Fire Model	epochArray	RESTful-CD++ indexed epoch data	234,664
entityWebWorker.js	array – Fire Model	aEntityArray	Storage array element of all entities	26,448
entityWebWorker.js	array – Fire Model	aGridDataArray	Storage array of all entities for grid data	84,644
entityWebWorker.js	array – Fire Model	aHeatmapArray	Storage array of all entities graphics	43,248
restfulResource.js	string – Occupancy Model	unzippedText	Uncompressed data – Fire model	28,064,688
riseWebWorker.js	array – Occupancy Model	dataInput	unzippedText split into line of text	27,772,072
riseWebWorker.js	array – Occupancy Model	arr	Elimination of duplicate lines of text	27,666,758
riseWebWorker.js	array – Occupancy Model	dataArray	RESTful-CD++ parsed and manipulated data	20,271,234
riseWebWorker.js	array – Occupancy Model	epochArray	RESTful-CD++ indexed epoch array data	21,773,466
entityWebWorker.js	array – Occupancy Model	aEntityArray	Storage array element of all entities	21,773,466
entityWebWorker.js	array – Occupancy Model	aGridDataArray	Storage array of all entities for grid data	84,644
entityWebWorker.js	array – Occupancy Model	aGridDataArray	Storage array of all entities for grid data	84,644

Table 3. Browser's Dynamically Generated Object Sizes.

5.2 Modern Web Technologies

Instructions to the client browser are provided by HTML versioning tags, such as those used in HTML4 which allowed the browser to compensate for language augmentation, language restrictions, language clarification, or language incompatibility. The new HTML5 doctype is now simplistic in implementation and backwards compatible to Internet Explorer 6, allowing the full range of dynamic and practical application development [W3C12 – HTLM5].

In the evaluation of the initial prototype entity simulation proposed by MÄk for a web-based client application development, certain discrepancies for HTML5 were initially discovered and corrected. These items are discussed below.

5.2.1 HTML Versioning

Instructions to the web browser about what version of HTML the page is written in is done by the *<!DOCTYPE html>* declaration. In order for the browsers to render the content correctly, the proper declaration is required. HTML5 is not based on Standard Generalized Markup Language (SGML), unlike HTML 4.01, which requires the Document Type Definition (DTD) declaration to specify the rules of the markup language. Usually the transitional DTD declaration is used with older browsers that can't very well understand CSS. As we are dealing with modern browsers, using HTML5 and CSS, it seemed contradictory to the requirement for more modern browsers with WebSocket and web worker capability. Figure 29 shows the original HTML4.1 coding that refers to the transitional DTD declaration instead of the simpler HTML5 declaration of *<!DOCTYPE html>* which is backward compatible with common parsing for older versions of HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
```

Figure 29. XHTML 1.0 Transitional DTD Declaration.

5.2.2 Downlevel-Revealed Conditional Comment

The second step in the modification of the MÄk entity simulation into a RESTful-CD++ distributed simulation interface was an examination of the technologies that have allowed the development of the WebVLC protocol and their implementation in the WebVLC simulation.

Examination of the web browser technology revealed that Microsoft had a later development cycle than Chrome, Firefox or Safari and the initial MÄk WebVLC prototype entity application was hardcoded as a conditional comment against all Microsoft Internet Explorer versions rather than as a browser's capability to handle HTML5 WebSockets or web workers. This conditional comment is shown in the JavaScript code of Figure 30.

```
<!--[if IE]>
        <div id="IEwarning">this browser does not support the current WebSocket
        protocol.  Please use a current version of Chrome or Firefox to view the live
        demo.</div>
<![endif]-->
```

Figure 30. Downlevel-Revealed Conditional Comment.

This code was modified into an unobtrusive JavaScript function that checked for the browser's ability to inherently handle WebSockets and the web workers compatibility. This function was called by the Dojo event handler for initialization after the map onload function and tests the capability of the browser rather than just whether Internet Explorer was used. The results of the conditional populate a success string into the DOM text box for the user's information. This is demonstrated by the JavaScript code snippet of Figure 31.

```
function browser_type()
{
        var browser = navigator.appName;
         var myWebWorker = webWorkerSupport();

         if ( window.WebSocket && typeof myWebWorker !== 'undefined' )
```

```
            document.getElementById( 'browsersuccess' ).innerHTML =
            '<p>Using ' + browser + ' - with WebSockets and WebWorkers</p>';
        else
            document.getElementById( 'browsersuccess' ).innerHTML =
            '<p>Please use a different version for a browser. Incompatible with
            WebWorkers or WebSockets.</p>';

        //Reset any text that may have been in the file path testbox
        document.getElementById( 'files' ).value = "";
}
```

Figure 31. Functional HTML5 Browser Capability Conditional

5.2.3 Webpage Refactoring

The original main html webpage developed by MÄk contained both HTML and JavaScript code. It is our belief in modularity, encapsulation and conciseness that it would be better served if the HTML web page containing HTML and the JavaScript functionality was exported to separate files we believe that having unobtrusive JavaScript allows the separation of content from the behaviour. This was accomplished and the HTML web page was renamed CDppEntitySim.html upon refactoring while the JavaScript files were further separated into window object JavaScript and non window object JavaScript files, named windowUtilities.js and utilities.js respectively.

In order to provide a user friendly interface, extra DOM objects were created and embedded within the CDppEntitySim.html file. The entity simulation was modified for a tabbed pane for the user to interact or examine entity data input and output as the simulation is running. A horizontal scrollbar and text edit boxes were customized to allow dynamic execution for the RISE resource results file, depending on the internal time event periods of the log file, its output value and output port. A data grid was embedded into one of the tabbed pane allowing examination and presentation of individual entity data to the user. A combo box was inserted into the HTML table to allow the user to select whether a DCD++ or a Lorez RESTful-CD++ resource file will be accessed. A start / stop command button was also introduced to allow the RESTful-CD++ log file to be controlled by the user in its population of data to the WebLVC server. This is shown in Figure 32.

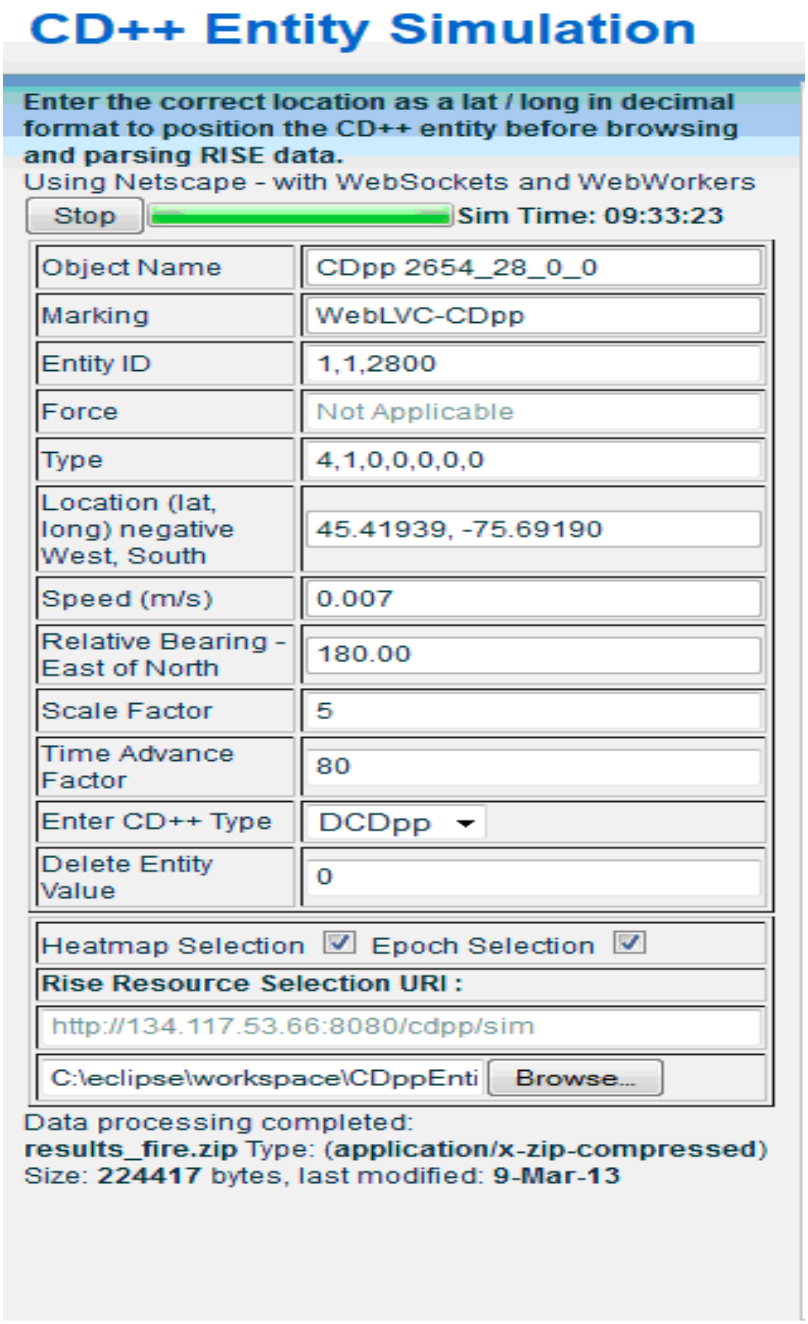

Figure 32. CDppEntitySim.html Left Pane

The CDppEntitySim.html webpage is composed of the following elements and is described starting from the upper left of the webpage:

a. Label: The title of the webpage;

b. Label: Information help string that reminds users to enter the origin location prior to selecting the start command button as all calculations are based on the initial position as the Cell-DEVS grid cell origin;
c. Label: Text informing the user of the browser capability with respect to WebSockets and web workers;
d. Command Button: Commences / terminates the RESTful-CD++ data interface with the WebLVC server;
e. Progress Bar: Indicates the processing progress of the RESTful-CD++ log file;
f. Label: Indicates the time since the webpage was loaded;
g. Table: Composed of labels, text boxes and combo boxes having the following identities:
    i. Object Name: The generated entity name composed of a string, a random number, and the RESTful-CD++ grid cell coordinates;
    ii. Marking: Character set and the string of characters used to provide display information for the entity;
    iii. Entity Identifier: Generated unique identifier of each new entity composed of a triplet site id, host id, and entity number for identification. Note that the entity number is limited to five digits by the WebLVC server;
    iv. Force: Enumeration identifier distinguishing the different teams or sides in an exercise
    v. Type: Enumeration to provide each entity as a object classification with an unique identifier in order to provide the minimum set of attributes needed to visualize an object in a distributed simulation;
    vi. Location: Latitude, longitude and geocentric position (if present) of the entity;
    vii. Speed: Calculated line of advance of the entity from the Cell-DEVS grid cell origin;
    viii. Relative Bearing: Calculated line of bearing of the entity from the Cell-DEVS grid cell origin;
    ix. Scale Factor: Numerated value of the Entity Scale Factor horizontal slider which represents the scaling between the RESTful-CD++ grid cell coordinates

as projected on to the ESRI ARCGIS map. Used to create the latitude and longitude spacing of the Cell-DEVS grid cell coordinates;

x. Time Advance Factor: Numerated value of the RISE Scale Factor horizontal slider which represents the scaling between the RESTful-CD++ epoch events and allows the speeding up or slowing down of the events recorded in the RESTful-CD++ log file;

xi. CD++ Type: Combo box that allows selection between the DCD++ and Lopez formats generated by the RESTful-CD++ server; and

xii. Delete Entity Value: Value associated with the MA extension file which will cause a Cell-DEVS grid cell coordinate to change from populated to unpopulated and further means the deletion of the entity from the WebLVC distributed simulation; and

xiii. Cell-DEVS Output Port: Port associated with the MA extension file rules which specify the final output port of the Cell-DEVS simulation execution.

h. Table: Composed of text boxes, check boxes and a command button having the following identities:

i. Heatmap Selection: When selected enables the generation of heatmap graphics for local visualization improvement;

ii. Epoch Selection: When selected enables epoch generation of entities and graphic rather than non-epoch generation;

iii. RISE Location URI: Pre-populated text box that allows the user to enter the URI of RISE;

iv. File Location: Text box that is populated with the file path of the user selected file; and

v. Browse Button: Command button that is used to produce an open file dialog for file selection and decompress of the RESTful-CD++ resource result file.

i. Label: Generated file attribute string that is dependent on the selected RISE resource results file;

j. Tabbed Pane: Composed of three panes that present useful information in both graphical and textual formats as follows; and

i. Map: User selected map that is defaulted to the ESRI Imagery World 2D map and contains the visual representation of the entities superimposed on top of the map as shown in Figure 33;

ii. RISE Webpage: Embedded webpage;

iii. Processed Data: Detailed tabulated entity information created for each RESTful-CD++ grid cell coordinates after being processed for entity generation as shown in Figure 34; and

iv. Sequence Calls: Used for debugging to determine the sequence of events for the web workers as the IDE is registered for the web page global space and not the web workers space. Allows the data view of the sequence of messages passed in the execution of the client application.

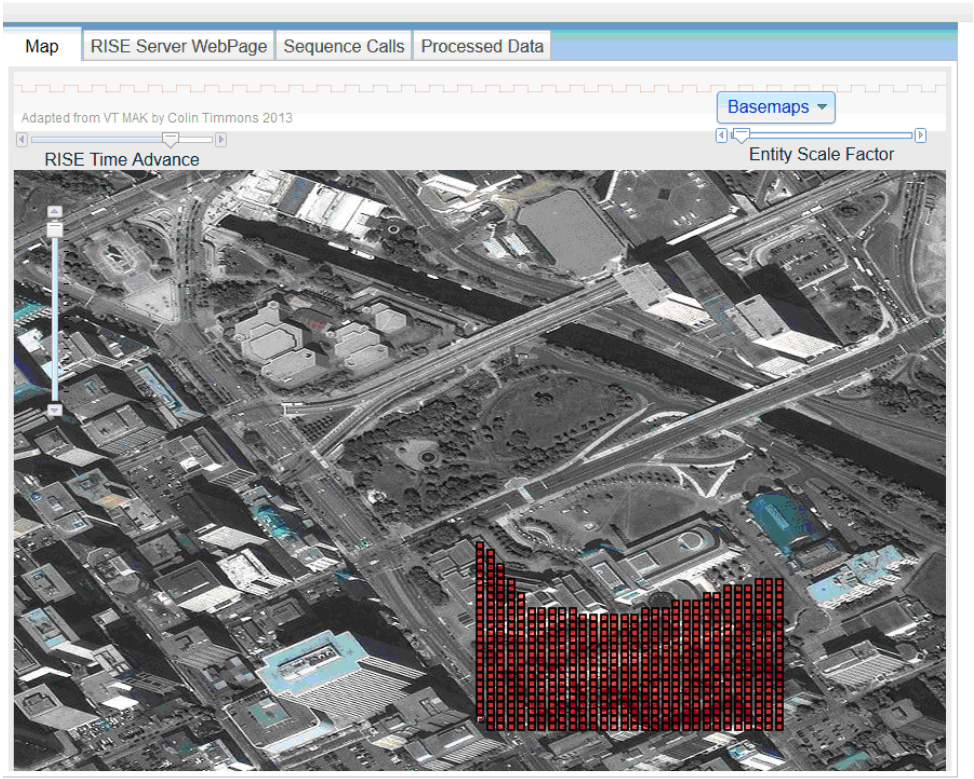

Figure 33. Map View of the CDppEntity Simulation.

Figure 34 demonstrates the data grid view of the RESTful-CD++ generated entities on the processed data tab pane. Each row represents a single entity and each column represents the particular information of interest of the entity. The index

number represents the line number of the data log file after all duplicated line of text are eliminated. The cell column represents the Cell-DEVS grid cell in a (x, y, z) coordinate system. The simulation time is the time since commencement of the thin client. The RISE time column represents the indexed number's event time gained from the data log file. The layer column represents the graphical layer the graphic is placed in while the value column represents the grid cell's value. The position, bearing and distance represent the calculate values of the grid cell based on the grid cell origin's (0,0,0) latitude and longitude as selected by the user.

Index	Cell	Sim Time	RISE Time	Layer	Value	Position	Brg	Distance
1797	28 0 0	00:17:23:000	07:09:53:405	8	430.890 36	45.41939 -75.69190	180.00	224.000
1796	29 1 0	00:17:18:000	06:58:23:639	7	419.377 58	45.41932 -75.69180	178.02	232.138
1795	27 0 0	00:17:11:000	06:56:36:563	7	417.592 98	45.41946 -75.69190	180.00	216.000
1794	28 1 0	00:16:59:000	06:45:05:797	7	406.080 2	45.41939 -75.69180	177.95	224.143
1793	26 0 0	00:16:59:000	06:43:18:722	7	404.295 61	45.41953 -75.69190	180.00	208.000
1792	29 2 0	00:16:53:000	06:33:34:031	7	394.567 42	45.41932 -75.69170	176.05	232.551
1791	27 1 0	00:16:42:000	06:31:47:955	7	392.782 83	45.41946 -75.69180	177.88	216.148
1790	25 0 0	00:16:42:000	06:29:60:880	7	390.998 23	45.41960 -75.69190	180.00	200.000
1789	28 2 0	00:16:33:000	06:20:16:189	7	381.270 05	45.41939 -75.69170	175.91	224.571
1788	26 1 0	00:16:33:000	06:18:29:114	7	379.485 45	45.41953 -75.69180	177.80	208.154
1787	24 0 0	00:16:25:000	06:16:42:038	7	377.700 86	45.41968 -75.69190	180.00	192.000
1786	29 3 0	00:16:18:000	06:08:45:423	7	369.757 27	45.41932 -75.69159	174.09	233.238

Figure 34. Data Grid View of the RESTful-CD++ Entities.

5.3 RESTful-CD++ Embedded Interface

RESTful-CD++ provides the results of its simulation modeling as a compressed zip file accessible by the user. Based on premise that the file would be accessed, read and manipulated into data readable by the WebLVC server, investigation was conducted into providing non-human interaction processing sub-routines that would provide the following procedures:

a. read a user input selection field;
b. automatically download the requested resource;
c. access the downloaded file from the file system;
d. uncompress the resource file;
e. identify the uncompressed simulation log file;
f. extract the uncompressed log file;
g. parse the uncompressed load file;
h. extract the relevant data objects; and
i. store the relevant data objects for future manipulation or retrieval.

Initially, the DCD++ fire model was used as the means to prototype the JavaScript code necessary to accomplish this tasking. The JavaScript code that accomplished these procedures was developed in the handleFileSel function of the windowUtilities.js file using the open source bitjs library [Bit13].

5.3.1 Cross-Domain Resource Sharing

Two implementations of RISE are hosted as a development machine located within the Virtual Simulation (VSIM) building and also as a computer resource server within the Mackenzie Building at Carleton University; both allow remote access. Singh as his PhD dissertation is remodelling and versioning the baseline of the RESTful-CD++. With three possible platforms, the compliance of CORS dependency based on the server location and the client-side JavaScript security implementation preventing file access for storage, it was decided not to attempt the modification of the request and response headers of RISE for CORS relaxation.

As the RESTful-CD++ provides multiple compressed files, it was necessary to be able to access the RESTful-CD++ services both as a viewable webpage for user selection and to be able to accept the different resources as dependent on the URI selected. Initially, CORS implementation was attempted in JavaScript in order to accept the compressed file. However, the response code of RESTful-CD++ demonstrated that RISE has not

implemented CORS through the Access-Control-Allow-Credentials specification as shown in **Error! Reference source not found.**. Accommodation for the access to the zip file could be andled through a hosting server. However, this would defeat the portability and the mash-up of services available to the client browser and require the refactoring of the RISE.

Figure 35. RISE Response Header.

Manual user selection was thus determined as the best option as it removed the CORS problem and also allowed the user to generate state information based on the selection of the URI of the RISE. User selection of the URI also permitted individual selection of different RESTful-CD++ resources which also aiding in the validity testing as a RISE distributed simulation interface. It was assumed that the software platform of the user may not have the ability to directly decompress the file or the knowledge of which file was pertinent and needed, so the decompressing of the RESTful-CD++ resource results file was directly incorporated into the client side.

The CDppEntity Simulation, as show in Figure 36, pre-populates the main web address for the RESTful-CD++ server. Upon the user submitting the address by changing focus, the

application displays the RESTful-CD++ web page on the CDppEntity Simulator's tabbed pane shown in Figure 37. URIs of the RESTful-CD++ server can therefore be followed while remaining in the iframe.

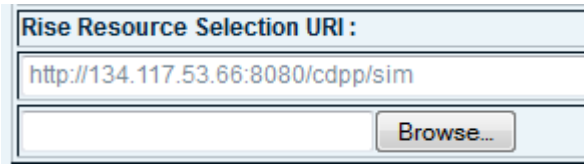

Figure 36. RESTful-CD++ Server Resource Selection.

Figure 37. RISE Embedded Tab Pane.

5.3.2 Decompressing the RESTful-CD++ Resource Results File

The RESTful-CD++ zip file, upon a user's selection of the correct file through the browse command button, was uncompressed and the files within the compressed file examined. To provide extra information to the user about the zipped file, the properties, size and date were analysed from the file attributes and displayed for the user to ensure the correct file was being extracted. This is shown in Figure 38.

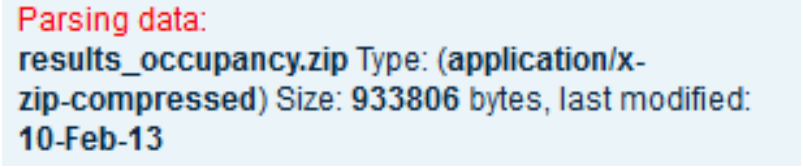

Figure 38. RESTful-CD++ File Properties.

The inflating of the RESTful-CD++ file was accomplished by the use of an open source JavaScript multi-threaded library file called bitjs [Bit13]. Since these method calls were incorporated into the CDppEntity Simulator, no unincorporated third party software or tools were necessary. The method calls of the file selection handler and decompressing calls can be examined in the handleFileSel function of the windowUtilities.js file.

The compressed file was inflated into an ArrayBuffer, examined for valid data text files in either a DCD++ or Lopez format, and populated into a string. This string was manipulated so that an array was created which housed elements that were representative of each line of the log output of the CD++ simulation engine. The data array was then passed to the web worker and evaluated to ensure that no two elements of the array were identical, as any duplicate would not allow a unique entity generation. Because of duplicability in the lines of output within the log file, a function was created to eliminate the duplicate number of output lines and return an array of unique elements. For example, as shown in Table 2 for the RESTful-CD++ DCD++ fire model, the number of duplicates entries was significant at 16.9% while the change in byte size of the original array was 20.8% when using the getObjectSize function. The Lopez output from the CD++ simulation engine contains only output Y messages with initially a small number of duplicate lines.

	Number of lines	Byte Size
Original number of entries in log file	45975	6,394,216
Remaining entries after duplication elimination	38190	5,066,610
Percentage of duplicates eliminated	16.9%	20.8%

Table 4. Fire Model Log File Duplicates Comparison.

Additionally, the elimination of the duplicates provided sequential ordering, reduced array size, and resolved data conflicts in the interface with the WebLVC interface. Further containers were created as sub-elements of the main array in order to provide indexing capability, and data retrieval. Upon completion of the parsing of the main array into subsequent data arrays, the main array was emptied for garbage collection and stack space.

5.4 Modification of the Entity Simulation

The entity simulation originally prototyped by MÄk, was used as the basis to provide a dual implementation as a RESTful-CD++ distributed simulation interface to the WebVLC server, and a RESTful-CD++ visualization tool. The MÄk entity simulation began as a starting point for understanding and implementing the WebVLC protocol that is presently being developed and possibly standardized in the future.

Vrlink.js, as supplied by MÄk, is the main software interface allowing information transfer between the WebLVC server and web-based client applications such as the CDppEntity Simulation. This file is used to provide the reflected entity data from the WebLVC server into the client applications for distributed simulations. The serialization of the data, however, is limited to DIS or RPR-FOM 1.0 format as shown in the brief code listing of vrlink.js of Figure 39.

```
  /**Fills out a WebLVC AttributeUpdate message, based on contents of
entityStateRepository.
    @method fillOut
    @param {EntityStateRepository} stateRep
    @param {EntityStateRepository} asSeenByRemote
    @return {Object} WebLVC AttributeUpdate message**/
    fillOut : function( stateRep, asSeenByRemote, attributeUpdate ) {
      attributeUpdate.AccelerationVector = stateRep.getAcceleration();
      attributeUpdate.AngularVelocity = stateRep.getRotationalVelocity();
      attributeUpdate.DamageState = stateRep.damageState;
      attributeUpdate.DeadReckoningAlgorithm = stateRep.algorithm;
      attributeUpdate.EntityIdentifier = stateRep.entityIdentifier;
      attributeUpdate.EntityType = stateRep.entityType;
      attributeUpdate.ForceIdentifier = stateRep.forceIdentifier;
      attributeUpdate.Marking = stateRep.marking;
      attributeUpdate.MessageKind = mak.MessageKindEnum.AttributeUpdate;
      attributeUpdate.ObjectName = stateRep.objectName;
      attributeUpdate.ObjectType = stateRep.entityType;
      attributeUpdate.Orientation = stateRep.getOrientation();
      attributeUpdate.VelocityVector = stateRep.getVelocity();
      attributeUpdate.WorldLocation = stateRep.getLocation();
    }
  };
```

Figure 39. Serialization Method of Vrlink.js.

As this serialization function is presently incorporated across all web-based MÄk test and prototype client applications for distributed simulation, it was decided not to refactor or modify the vrlink.js code. Retesting and modification of the present prototype simulations presented by MÄk was well beyond the scope of this thesis as each developed and new client application would also require a new vrlink.js library file. Presently MÄk is concurrently developing the WebLVC server for HLA interfacing and mapping the WebLVC-to-HLA and HLA-to-WebLVC interactions. As such, a limitation was accepted that the CDppEntity Simulation would continue using the present DIS format and accept the RPR-FOM 1.0 entity generation.

Continuing from this limitation, the MÄk entity simulation was evaluated in order to see how the application produced data for interfacing to the WebLVC server. It was discovered that the following actions were performed by the MÄk entity simulation.

a. WebSocket Generation: The prototyped entity simulation created a WebSocket immediately upon the map loading and was directly linked to the main JavaScript constructor for the entity;
b. Entity simulation Generation: The main JavaScript constructor was initialized with a static client name, object name, entity type, force identifier latitude, longitude, pitch, roll, heading and speed. These are only a few of all the required attributes;
c. VehicleSim Generation: The main webpage initialized an entity simulation object and also initialized a vehicleSim object to handle changes in the initial location, speed, heading and rotational changes of the entity as it progressed through its lifecycle. However, there was only one vehicleSim object created, meaning that only that entity's updated parameters could at any one time be published to the WebLVC server;
d. Entity Display Selection: Originally, MÄk allowed only specific entities to have representation as an image such as an F-18 aircraft through a switch conditional based on the value of the entity type or representation as a graphic dot; and
e. Timing Advancement: The vehicleSim class was responsible for creating and advancing the time properties and updates of the single entity to the WebLVC server as a delta of 0.05 seconds.

Modification of the design of the original MÄk entity simulation required a complete rethinking. The following adaptations were created to add robustness:

a. Application Coordination: A new main class was created called CDppApplication that generated and controlled the resources and lifecycle available to the other classes that compose the CDppEntity Simulation. This would allow the generation, population and elimination of the RESTful-CD++ event data through the existence of the WebLVC distributed environment. Within the application class, the URI of the application was read to gain access to the user's input to the simulator interfacing parameters through the query string rather than at the creation of each entity;
b. Entity Resource Management: A MÄk modified entity class was refactored to allow multiple instantiations of the vehicleSim class and thus multiple entity generation for the WebLVC server;
c. WebSocket Decoupling: The WebSocket was decoupled from the original MÄk entity simulation in order to allow the multiple generated entities to use the same port for the common WebSocket. If this was not completed, each entity would have had an assigned port and a new WebSocket for data transmission and reception. With the WebLVC server expecting a WebSocket connection per application, this would have enticed problems;
d. Initializer Modification: The MÄk initializer object was modified to handle the different new dynamic latitude and longitude coordinates for each RESTful-CD++ grid cell coordinate as detailed in the RESTful-CD++ resource results log file. Also, this class was modified for the generated bearing and distance of each entity from the original user defined initial latitude and longitude positional coordinates;
e. RESTful-CD++ Resource Coordination: A class was created to handle the specific data parameters required to interface the RESTful-CD++ resource file and to transpose the data into a format acceptable for interfacing with the WebLVC server and visual display;
f. Visualization Creation: The original MÄk images selection was through a switch conditional while symbol generation was incorporated into each method. This was refactored in a visualization JavaScript file allowing modularity in code

maintenance. A heatmap selection capability was developed to aid in the visualization of the RESTful-CD++ data; and

g. Utilities Library Creation: The MÄk JavaScript located in the main webpage was moved to a windowUtilities.js JavaScript file to provide unobtrusive window object JavaScript. Other common window object methods created for the CDppEntity Simulation main UI thread were housed within this file. Non window object and document functions were created in a utilities.js JavaScript file for library uses by all files including web workers; and
h. CDppEntity Multi-Threading: The CDppEntity Simulation incorporated multi-threading through the use of web workers to provide a stable responsive client application rather than relying on the single UI thread to execute all required processes in order to create the mash-up of services.

5.4.1 Application Coordination

The WebLVC server requirement follows the DIS Entity Identifier 3-tuple of site identifier, host identifier and entity number to uniquely identify entities in a distributed simulation. The CDppApplication JavaScript object was developed to generate and to control default settings for the CDppEntity Simulation client application. Public member variables hold the site identifier, the host identifier and a pseudo-random number generation along with the user specified latitude and longitude positional coordinates for the initial positioning. Since the application name is also common, the CDppApplication also houses the application's client name and singleton method instances allowing for changes in the client name or location via a query string input to the URI of the CDppEntity Simulation as shown in Figure 40.

192.168.0.11:8080/CDppEntitySim.html?objectName=comp1&hostId=2&marking=fire

Figure 40. User Supplied Query String Parameters.

As the main instance object when the application is commenced, the CDppApplication object holds the references for the common WebSocket, a reflected entity list, a marshalled

MÄk storage object, for the incoming WebLVC server data, and a method to ensure that clean-up, if required, is executed upon stopping the simulation.

5.4.2 Entity Resource Management

A CDppEntitySim.js file was created based on the MÄk entitySim.js file but was modified to allow multiple entities to be dynamically created with reference to the RESTful-CD++ cell grid coordinate system. Hence, each dynamically created entity would have a different location position relative to the initial latitude and longitude positioning defined by the user. This initial user defined position coordinate was the RESTful-CD++ origin coordinates (0, 0, 0) and was defaulted to Ottawa's latitude and longitude with a zero third element. Since the entities would be offset by some distance from the origin, a bearing and distance calculation was also created and stored within the CDppEntitySim object. Initializer data and the entity's timer were tracked and stored into an array for validation of entity creation and to ensure instance verification that the RESTful-CD++ grid cell is not recreated when it is already present as the RESTful-CD++ log file is processed. As the MÄk prototype considered only one entity generated per application and instantiated the entity upon loading, the start / stop entity creation functionality was refactored to the windowUtilities.js file to provide a calling method for multiple entity creation and publication.

5.4.3 WebSocket Decoupling

As stated above, multiple entities could exist and therefore a common WebSocket was needed to handle the data transmission and reception for the WebLVC server. Originally in the one entity version proposed by MÄk, a WebSocket was created immediately upon the loading of the web-based client application for communication with the WebLVC server. In order to resolve the multiple entity condition, the WebSocket creation was decoupled from the web-based client side entity's creation and moved into the initialization domain. The CDppEntity object would then use the CDppApplication's common WebSocket to transmit

its data to the base object for publishing. Since JavaScript is inherently single threaded in this context, synchronization primitives are not required for callback routines.

It would have been more expedient to host the WebSocket in the web worker thread space; however, this was not possible and the WebSocket was kept in the UI thread space. Though the XMLHttpRequest method is allowed through the TCP/IP topology, the web workers were restricted and would throw an error trying to create a WebSocket. Similarly, though the common WebSocket could be shared with the other entity classes and data could be passed between the memory spaces, the WebSocket object was not permitted to be passed to the web workers after the WebSocket creation.

5.4.4 Initializer Modification

A CDppInitializer object was developed to store the default settings for the entity as member variables. These member variables contained parameters such as the entity's object name, latitude, longitude, altitude, heading, speed, turn radius, roll, pitch, type in accordance with the RPR-FOM 1.0, and the RESTful-CD++ grid cell coordinate. Continuing with MÄk's use of purl.js, the parseURL function was used to handle query strings when the HTML web page was loaded, but was refactored for new entity object names, and new entity numbers.

Each entity must be unique and thus the RESTful-CD++ grid cell position was originally appended to the client's generated random number. However, the entity number is limited to five digits when inputted into the WebLVC server and so only the cell coordinates were used. Though each grid cell position can be considered unique, it is not truly so if one considers the case were multiple similar applications were run simultaneously; thus, the requirement is generated to append the cell number along with the client application object name in order to generate an entity's unique name. Accepting the number field limitation of the WebLVC server, the CDppEntity Simulation produces the simulation application name and the 3D grid cell coordinates as the unique entity name.

5.4.5 RESTful-CD++ Resource Coordination

A new file was developed to handle the requirements of parsing the RESTful-CD++ resource file to obtain data for the WebLVC server. This file, restfulResource.js, contains all the methods needed to control and handle messages passed to its child web workers. It transmits the visualization outputs as a WebLVC entity or generates the dynamic heatmap graphics and updates the Dojo data grid with the contact information of each RESTful-CD++ grid cell coordinate.

The restfulResource.js file is summarized as follows:

a. constructor: Create a new instantiation of the RestfulResource class;
b. createHeatmapEntity: Creates a layered heatmap entity for local display for the graphics based on the value of the RESTful-CD++ grid cell coordinates;
c. createWeblvcEntity: Wrapper for creating a WebLVC entity by initializing a common utility function with appropriate data elements;
d. deleteEntity: Non-Epoch wrapper for deleting values in the arrays used for the heatmap, data grid store, and the locally held entity list. Once the data is deleted a WebLVC delete message is sent out to delete the entity from the WebLVC server and hence the simulation;
e. deleteWeblvcEntity: Wrapper method for creating a WebLVC delete message and removing the local to-be-deleted graphics from display;
f. initComms: Converts the decompressed log file string into line of event details and posts DOM initialized variable values as messages to the child web worker threads for commencement of the population of data to the WebLVC server;
g. initRestThread: Method for creating the web workers, handling exception errors from the web workers and message handling event listeners for the web workers;
h. initWeblvcEntity: Wrapper for creating a WebLVC entity by initializing class base parameters. Actioned via the message channel when data is to be populated via an RESTful-CD++ event;
i. popGridArray: Wrapper for populating an array containing the objects for display on the tabbed pane processed data grid view;

j. processRise: Processes the stored arrays at the required epoch, produces a layered graphic, creates the WebLVC entity, and populates the tabbed pane processed data fields. Used in the non-epoch selection;
k. setEpochData: Method to create an entity and then populate the RESTful-CD++ data message from the RiseWebWorker for the EntityWebWorker;
l. setGridData: Wrapper method to set the data array into the Dojo data grid for tabbed entity information viewing;
m. setHeatmap: Wrapper method to set the data array into the heatmap layer for visualization graphics;
n. setObjData: Debugging function to set Dojo content pane with messages passed between threads;
o. stepTime: Called by the timer which is based on the RESTful-CD++ event times and is the method which calls the processRise for simulation events. This method also updates the progress bar which is based on the number of discrete events epochs of the RESTful-CD++ log file;
p. termRestThread: Function to terminate the worker threads and allow garbage collection of their memory allocations; and
q. updateTab: Wrapper function used to set the data array into the dojo data grid for viewing on the tabbed pane of the processed data.

A RestFulResource object is created to represent a class structure for handling the RESTful-CD++ resource file. The file access method of the windowUtilities.js file which accessed and processed the compress file, stores the event data into a RestfulResource string handler. This string is then further manipulated by its carriage line feed characters into an array of elements where each element represents a line of the Cell-DEVS simulation execution. Upon completion, the new data array is message posted to the RiseWebWorker thread for data parsing, extraction and processing. Along with the log event data array, other parameters are sent to the RiseWebWorker so that it may correctly extract the proper data. Such parameters are the user entered zero-population value array for the Cell-DEVS grid cell coordinates, the user selected combo box selection, the user selected final

output string, the client side name for generating the entity object name of each Cell-DEVS grid cell coordinate and the client side's dynamically generated random number.

The parsed data is stored and accessed once the user wishes to start the publication of the Cell-DEVS data and is discussed in Section 5.4.8 CDppEntity Multi-Threading. This is accomplished by selecting on the start command button which initiates a new WebLVC reflected entity list, clears the data grid, starts the simulation population clock, and commences the RESTful-CD++ epoch timer for continual processing of the data elements, creation of entities, transmission and display. Clicking the stop command button stops the processing, resets data fields, deletes the entities, and removes any local entity graphics.

5.4.6 Visualization Generation

Locally displayed within the CDppEntity Simulation is the ability to access different graphic layers and opacities which allows increments of colour to reflect upon the entity by the value of the RESTful-CD++ grid cell. This is specifically significant for displaying graphics such as the DCD++ fire model on high resolution maps whether the client application is connected via the WebSocket to the WebLVC server or not.

Modularity in code required the refactoring of MÄk supplied ESRI SimpleMarkerGraphic and graphic calls into modular function calls in accordance with the Don't Repeat Yourself (DRY) principle. Using the Single Responsibility Principle (SRP), the function calls created either a ESRI graphic or a SimpleMarkerSymbol for use by the CDppEntity Simulation to display the RESTful-CD++ data. The method calls visualBySymbol and visualByEntityType allow creation by different attributes such as colour, size, positional information and name.

The MÄk imageByEntityType function returned an image based on the entity type was refactored to be more selective in the image selection. The default originally coded function returned a Ford Contour Sedan as the image selected for the CDppEntity Simulation. Rather than using the second element of the DIS entity type array, the function was recoded to handle environmental entities as well as a Ford car for the land platforms and certain

aircraft for the air platforms. As the enumerations list is very lengthy with only a few of all the possible images present, it is understandable why the conditionals were not finalized. Unfortunately, the DIS Enumerations do not have specific enumerations for possible environmental entities like water for flooding, fire, or ground area for radioactive contamination and would be defaulted to the same graphic symbol under the other environmental entity enumeration.

5.4.7 Utilities Creation

Common to all the files is the utilities.js module file which houses common methods and functional objects used by the CDppEntity Simulation. The original MÄk supplied code is housed in a separate windowUtiliities.js file to ensure thread security with the web worker threads. Obtrusive JavaScript contained within the MÄk HTML webpage was relocated into the windowUtilities.js file in order to encapsulate the method calls from the user and to provide a separation of content.

MÄk provided a rich starting point for development and provided robust functional calls, though coded, not all were not all implemented and were left to assist a developer in modifying the MÄk supplied code into a viable entity simulation platform. The original functions contained in the entity simulation HTML webpage are now located in the windowUtilities.js file. The methods are listed as follows and denoted as whether modifications were required to develop the CDppEntity Simulation:

a. createBasemapGallery: Modified. Refactored to guarantee the loading the ESRI Imagery World 2D map first rather than the ESRI StreetMap World 2D map because if a delay occurs from the ESRI server, the ESRI StreetMap World 2D map becomes the default map even though the selection indicated by the original code called the ESRI Imagery World 2D map;
b. createEntityGraphic: Modified. Refactored to use the winsowUtilities.js function visualByEntityType and removed the orientation calculation of the entity;
c. createSpeedColor: Unmodified. Used to change the colour of the entity if the speed of the vehicle is over a certain threshold;

d. endWith: Unmodified. Used to check that a string ends with a certain number of characters;
e. init: Modified. Refactored to handle the CDppApplication object;
f. initF18: Modified to initEntity. Refactored to handle multiple instances of entity generation and storage of the entities created;
g. mapClickHandler: Modified. Refactoring to handle the user's initial latitude and longitude position, clears the processed data's tabbed data grid pane and entities array, and stops the CDppEntity Simulation;
h. mouseMove: Unmodified. Function handler to handle the mouse movement over the entity to get the entity's object name for display;
i. mouseOut: Unmodified. Function handler to handle the mouse movement away the entity to remove the entity's object name from display;
j. removeEntityGraphic. Unmodified. Removes the specific entity graphic from the map assuming the entity object name matches the published WebLVC entity attributes and object name;
k. startButton: Modified. Developed but not used in the prototype for initializing and populating the program for interaction with the WebLVC server. Commences the simulation timing, resets the processed data's tabbed pane data grid view, zeroizes and sets the range of the progress bar, and commences the RESTful-CD++ event timer;
l. stopButton: Modified. Function handler to handle when the user clicks the stop command button. Clears the entity generation timer, clears the simulation timer, clears the entities, and clears the data entered into the webpage text boxes;
m. unload: Unmodified. Used to remove any WebLVC reflected entities from the reflected entity list for client application when refreshed or unloaded;
n. updateEntityGraphic: Modified. Used to update the graphics used to represent the entity in the local viewer. Refactored to handle entity resize values and the visualization.js visualBySymbol function;
o. updateEntityGraphics: Unmodified. Timer method used to repaint the reflected entity from the WebLVC server;

p. updateGui: Modified. Updates the tableau of the left pane of the CDppEntity Simulation;
q. updateSize: Modified Function handler to handle the changing of the Entity Scale Factor slider bar and repopulate the value into the scale factor text box.

Other functions developed and added to the windowsUtilities.js file are as follows:

a. browserType: Used to determine the browser type and whether the browser is capable of handling WebSockets and web workers and outputs an information string to the user as to the results;
b. changeState: Function handler to handle user interaction with the start / stop command button and calls the setting of the zoom scale and setting of the map visible area
c. createFireOpacityLayer: Used to instantiate a fresh empty set of layers indexed above the map layer;
d. cursorClear: Used to return the frame cursor to the normal pointer;
e. cursorWait: Used to change the frame cursor to the wait state (busy) cursor;
f. getLocationAsObj: Used to return the initial latitude and longitude of the Cell-DEVS grid cell origin as defaulted or as provided by the user;
g. initEntity: Originally initF18. Creates a new entity for publishing to the WebLVC server and stores the entity information;
h. initFormFieldHandlers: Modular function to assign and initialize member control variables to the web page DOM objects and handlers;
i. handleFileSelect: Function handler to handle the selection of the desired compressed RESTful-CD++ file, the decompression of the RESTful-CD++ file, and the conversion of the returned data into a format acceptable for the string handler;
j. onError: Method used to write error text to the console output;
k. onLog: Method used to write log text to the console output;
l. removeAllGraphics: Used to remove all graphics from the map layer and removes all the added heatmap layers. It also resets the simulation progress bar to an initial position;

m. removeReflectedGraphic: Removes reflected entities published by the WebLVC server when the client application is stopped;
n. resizeEntity: Resizes the graphic depending on the user selection of the scale of the ESRI map selection. Returns the zoom factor needed to be provide a constant graphic size and spacing of the RESTful-CD++ grid cell coordinates;
o. retFireGradient: Returns a custom heatmap gradient colour with varying opacity for the heatmap layer;
p. riseDelay: Function handler to handle the changing epoch events of the RESTful-CD++ time advance slider bar;
q. setEntityType: Function to handle the user input selection of the type of entity to be generated;
r. setLocation: Method to convert the string of the parsed RESTful-CD++ grid cell coordinate to a LatLon object or to set the initial position as Ottawa's latitude and longitude;
s. unloaDCD++: Used to clear the client application map graphics and layers upon the client application unloading ad called by the Dojo event handler;
t. webWorkerSupport: Returns whether the browser can support web workers; and
u. zoomScale: Used to change the scaling and thus the scales view of the ESRI ARCGIS map in order to see the entities representing the RESTful-CD++ grid cell coordinates.

Developed non-window or document methods were written within the utilities.js file. This file was exported as a module and contains functions, prototype methods and objects. The following functions are listed and explained as follows:

a. arrayBufferToString: Function to convert a BLOB such as an ArrayBuffer into an indexable uint8Array such as a string;
b. compareArrays: Function that compares arrays for identicalness in the length and in the elements of the arrays;
c. clockTime: Formats an input value in seconds into a human readable format for display on the HTML webpage and for the processed data tabbed pane's data grid view;

d. clone: Reiterative method to copy an object, date, or array. While the this.slice(0) method returns a copy of an array, it will not allow the copying of whole objects with arrays;
e. createObjectName: Used to create an unique object name for transmission to the WebLVC server and is based on the RESTful-CD++ grid cell coordinates and the client application 's name and generated random number;
f. deleteObjLikeArray: Used to allow the deleting of a single element from an object. In JavaScript, this function deletes elements of an array while the delete command deletes only the properties of an object and not the elements leaving an array with undefined elements intermixed with defined elements;
g. divByZero: Used to prevent division by zero errors or singularities with tan / atan. In JavaScript, it prevents a Not A Number (NaN) return on division or returns 0 if the divisor is zero;
h. eliminateDuplicates: eliminates any duplicate elements found in an array;
i. extrapolate: Creates a new latitude and longitude object for the Cell-DEVS grid cell coordinate based on the initial grid cell origin of (0, 0, 0). The new object contains the latitude in decimal degrees, the longitude in decimal degrees and defaults to the radius of the grid cell number from the centre of the earth in metres;
j. getBearing: Determines the bearing between two latitude and longitude objects;
k. getCellSpace: Parses a string such as the Cell-DEVS grid cell coordinate string and returns a new numerated grid space identity object containing the x, y, and z axis coordinates;
l. getDistance: Determines the distance between two latitude and longitude objects;
m. getMemorySize: Testing function to determine the limits of the browser for memory size;
n. getObjectLength: Used to overcome JavaScript's limitation on returning the number of elements of an object. Similar to an associative array length command;
o. getObjectSize: Testing method to determine the byte count of a string, number, Boolean, or object. Implemented as a limited version of the sizeof method in C;
p. getStartTime: Function to return the start time of the population of data to the WebLVC server;

q. getTime: Parses a string such as a Cell-DEVS time element string and returns the number of seconds;
r. init: Function to initialize member variables;
s. isEmpty: Determines if the object or array is empty of data;
t. isGridIncluded: Used to determine if an object exists as an element of the data grid array which is used in the processed data's tabbed pane data grid view;
u. isObjIncluded: Function to determine if an element exists and if the value matches the required attribute of the function;
v. retFireLayer: Used to return the index for the heatmap gradient based on the value of the RESTful-CD++ grid cell;
w. searchString: Method to find the index of a user specified string in a supplied string;
x. setArrayLog: Function to store the sequence calls of the message posting from the web worker to the event listener of the web workers;
y. setEntityMsg: Method that creates and returns an entity message object base on the index parameter of the array; and
z. transMsgLog: Method to trace the sequence of activity of message passing as the debugger cannot see into the web worker memory or stack space.

The following objects that are used by the threads are listed and explained as follows:

a. GridCell: Numerated positional object created to store the Cell-DEVS grid cell coordinate;
b. LatLon: Numerated positional object created to store the latitude, longitude, and altitude of the RESTful-CD++ grid cell coordinate;
c. arrayMsg: Message object to store the RESTful-CD++ parsed log data into multiple named arrays within the object;
d. entityDelMsg: Message object to hold information about a single entity being deleted from the WebLVC server in the distributed simulation;
e. entityMsg: Message object to store the information about multiple entities being deleted from the WebLVC servers and is stored as multiple named arrays within the object;
f. epochMsg: Message object to store all the epoch information for a single entity;
g. gridMsg: Message object to store all the data grid information for a single entity;

h. heatmap: Message object to store all the heatmap information to create a single heatmap entity;
i. layeredValueMsg: Message object used to store multiple the RESTful-CD++ parsed log data in combination with the heatmap and map information for multiple entities;
j. riseMsg: Message object to encapsulate the RESTful-CD++ parsed log data for a single entity;
k. weblvcMsg: Message object to encapsulate the information used by the CDppEntity Simulation to display information on the left paned tableau; and
l. workMsg: Message object to encapsulate a command message and data parameters for message transmission on the message channels.

The following object prototype functions are checked for existence in case of namespace contamination and are used by the threads to manipulate data:

a. Array.prototyped.max: Returns the maximum value in the specified array;
b. Array.prototyped.min: Returns the minimum value in the specified array;
c. Array.prototyped.remove: Deletes the specified element of the array based on the index parameter without leaving an element of the array as undefined;
d. Number.prototype.toDeg: Converts the radian measure to degrees for any number;
e. Number.prototype.toRad: Converts the degree measure to radians for any number; and
f. Object.identical: Compares an object or elements of an array for identicalness.

5.4.8 CDppEntity Multi-Threading

As stated previously, the CDppEntity Simulation is composed in modular component, the unobtrusive JavaScript was kept as much as possible out of the CDppEntity HTML webpage and common global methods were accessed by the application. Window and document methods were written into a windowUtilities.js file while common non-window or document methods were assigned the utilities.js file. Methods particular to the RESTful-CD++ log file were housed in the RestfulResource.js file and the WebLVC simulated classes for the dynamically generated entities were housed in the cdppEntitySim.js. Retaining the

interface to the MÄk vrlink.js , the vehicleSim.js was only slightly modified to accommodate the RESTful-CD++ data entities.

Execution of the CDppEntity Simulation occurs upon the local or hosted web page being loaded by a compatible browser such as Firefox, Safari, or Internet Explorer 10. The client application was initially set up as a DIS environmental entity in accordance with the DIS application protocols [IEE12] and DIS Enumerations [DIS11]. The reason for modification with web worker threads is to allow users to interact with the web page form so that modification was permitted to the input fields and graphics without scripting non-responsiveness.

Moving away from the original prototype application developed by MÄk, the CDppEntity Simulation became multi-threaded in order to expedite the processing of data, without affecting the visualization of pre-processed Cell-DEVS data. Specifically, the RESTful-CD++ log file contains information on the state of the Cell-DEVS grid cells and the time event periods encompassing the transition of the data. As such, one can imagine at the extremes that the epoch periods may contain either a minimal amount of data, or a maximum amount of data. Conversely by examining the time period events, one can infer that there may be the same periodic/aperiodic events of small duration or large duration. Not knowing of the time epoch events makes the generation of the networked visualization extreme. In one case for small duration epoch events, the communication messaging may serve as a bottleneck and could foreseeably saturate the main UI thread and cause latency problems because of the rapidity of the data to be processed in such a short time period. In an opposite case, with a large time separation, the size of the data being transferred may interfere with a timely and correct operation because of the amount of data passed.

As JavaScript is originally single threaded, the main UI thread becomes slow and sleep or wait functions would terminate or interfere with the user's use of the application. In order to prevent the main thread from appearing non-responsive, time-consuming tasks were offloaded into HTML5 web worker threads. Each thread, as required, was located in a separate file allowing it to be self-contained with its own memory space.

Without the web worker threads, all data would have to be housed within the main application UI thread space, or within a file which would be read. With the JavaScript security policy which allows the reading of files but not the ability to create them, the RESTful-CD++ data had to be stored locally within the application. As the RESTful-CD++ compression file could potentially be limitless in size, this posed a problem with the UI thread stack size with the storing data into an array, parsing the data for valid information, retaining that information, and configuring which data was important at the time event periods.

With web worker threads, the memory space of each thread could be utilized to create a platform to house the RESTful-CD++ data, and extract the validity of state events for separate time division periods. However, coordination between the threads was paramount as the transmission of data and its subsequent processing attest to the importance of having the data present before computational processing. The message channels that the web worker communicates with its parent thread allow the transmission of strings and the primitive JavaScript objects as a JSON object. Web workers also have the ability to import scripts into their memory space allowing modularity in code by having the utilities.js file common to all threads. The ability of the riseWebWorker.js file to parse and correctly identify valid data is dependent on the RESTful-CD++ resource file structure remaining static.

Table 5 illustrates the message event commands transmitted by the UI and web worker threads. Each web worker thread is designed to provide an update to its status to the UI thread in order to provide reliability in the reception, processing and transmission capability when a message command is sent from the UI thread. Thus, for example, when the UI thread wants to initialize the data variables in the web worker threads for the user selection of the combo box, it would post a message to the respective individual threads with the message command setComboSel. The web worker threads, upon receipt of the message, would initialize their assigned variables to the value of the data passed and would respond with a consoleLog command message to the UI thread with a success state string.

The UI thread, upon receipt of the message, would log the message event data to the console for the user to see when activated.

Upon initialization of the threads and its subsequent loading of data, the threads process the data under control of the messages that are passed. The sequence of messages required to extract and populate data for entity creation and display are shown in Figure 23 and Figure 24. The threads, in order to process data correctly rely on the utilities.js library module that is common to all threads and is loaded by the web worker importScript command.

The RiseWebWorker is responsible for the extraction and processing of the Cell-DEVS log file and determines what data is redundant, what the time period events exists and pre-populating the CDppEntity Simulation with the initial data value requirements. The criteria for extracting the data is dependent on the user's selected control inputs that dictate the final output port and the non-populated value of each Cell-DEVS grid cell coordinate. Upon a successful condition, each line of the Cell-DEVS log file is broken into elements and the value is extracted and manipulated if required for each element.

Message Command	Issued by Thread			Reason
	CDppEntity	Rise Web Worker	Entity Web Worker	
consoleLog		✓	✓	Used to log debug information to the console.
getArraySourceLength	✓			Get the length of the main array's source element.
getArrayDiffTimeLength	✓			Get the length of the main array's time difference element.
getTimeDifference	✓	✓		Get the difference in events for the CDppEntity Simulation timer.
parseArray	✓	✓		Populate the RESTful-CD++ data and store the initial latitude longitude, and scale.

setClientInfo	✓		✓	Provide the application name and random number.
setComboSel	✓	✓	✓	Set the user's selection on the format of the RESTful-CD++ compressed file.
setEntityTableData			✓	Return the array of entities for deletion to the UI thread.
getEpochArrayList	✓	✓		Get the next element of the array.
setEpochData	✓		✓	Received epoch data from RiseWebWorker and need to repopulate it with simulation data for the EntityWebWorker.
setGridList			✓	Returns the array of data grid entities to the UI thread.
setHeatmap	✓	✓	✓	Informing the Entity WebWorker of the user's heatmap selection.
setHeapmapList			✓	Returns the array of heatmap entities to the UI thread.
start	✓	✓	✓	Initialize default values before unzipping files.

Table 5. Threaded Message Channel Event Commands.

The functions comprising the RiseWebWorker are documented as follows:

a. calculateDiffTime: Calculates the epoch time periods based on the Cell-DEVS log file and stores both the time of the event occurring and the difference in time between the time event periods;
b. createEpochEntityList: Function that creates an epoch period array comprising of the parsed DEV-CD++ data elements for each Cell-DEVS grid cell coordinate having an indicated state change;
c. getEpochArrayList: Function to return the indexed element of the epoch array;
d. init: function to initialize member variables;
e. messageHandler: Function that allows the communication messages to pass from the parent thread to this child thread. Based on the message received, the event

handler calls the appropriate functions and transmits data or signals to the parent thread;

f. parseArray: Main method that calls for the removal duplicate information from the data array which contains the Cell-DEVS data, creates and extrudes the white space separated line elements of the Cell-DEVS data, and calls for the creation of the bearing, speed and distance of the grid cell coordinates from the origin;
g. processDataForDisplay: Creates a series of layered elements for the layers value message which is used for non-epoch user selection;
h. setEpochArrayList: Function to wrap and store the created epoch entity list; and
i. transMsgLogger: Function to debug and help trace the web worker calls as the debugger cannot see the web worker thread space and the web worker cannot access the console.

The EntityWebWorker is responsible for data during each epoch event period. Its main purpose is to search the arrays and determine whether the positional value of the Cell-DEVS grid cell coordinate matches the user selected output value and remove the Cell-DEVS grid cell from being a WebLVC entity, if already present. If not present and the value is not of the deletion requirement indicating an empty cell, the EntityWebWorker will package the required information to create one into its entity array. Similarly it will perform the action on the data grid array and the heatmap array. Upon an epoch event request, the resource will send the newly constructed events array to its parent thread for processing on the heatmap, on data grids and for the deletion message creation.

The functions comprising the EntityWebWorker are summarized as follows:

a. createEntityDeletion List: Function to test and insert an epoch event array of entity identifiers into the end of the main deletion list for tracking of the entities deleted. Removes the entity from the container holding the entities;
b. createGridDeletionList: Function to test and remove a epoch event array of data grid information from the main data grid array ;
c. createGridDeletionList: Function to test and remove a epoch event array of heatmap positional information from the main heatmap array;

d. getHeatMapObjSize: Debugging function used to determine if the web worker memory space was significantly reduced;
j. init: function to initialize member variables;
k. messageHandler: Function that allows the communication messages to pass from the parent thread to this child thread. Based on the message received, the event handler calls the appropriate functions and transmits data or signals to the parent thread;
e. setEpochData: Function that accepts pre-populated data from its parent thread and determined the validity of the data for re-population into the local arrays of object. User selected conditions are used by the function to track data and to remove data from the arrays if conditions warrant; and
f. transMsgLogger: Function to help trace the web worker calls as the debugger cannot see the web worker and the web worker cannot access the console.

5.5 JSON Communication Protocols

In dealing with a RESTful WS and the transmission of data through the GET, PUT, DELETE and POST methods of HTTP, and transmission reception object was developed in JavaScript. The purpose was to be able to publish and consume JSON data from the RISE server if an interface wrapper was created in the future to transmit JSON in concert with XML or HTML across domains. The following methods were created as public methods of the txrx object:

a. checkJSONP: Function to check for the correctness of the JSON data;
b. create: PUT method for a JSON object using XMLHttpRequest;
c. getJSONP: GET method for a JSON object and appends it as a script to the head of the HTML file;
d. onLog: Warning function that outputs error messages to the console log;
e. remove: DELETE method for a JSON object using XMLHttpRequest;
f. update: POST method for a JSON object using XMLHttpRequest; and

g. xmlhttp: a member variable whose function is to return the correct XMLHTTP object depending on the type of browser.

5.6 Testing and Performance

In testing a web-based client application, the main criteria for a user are responsiveness, usability, and correctness. In order to get correctness, unit testing was continuously performed on the common methods to ensure validity to the testing, the development of code and sanity checks. The unit tests were developed prior to code development and were continually run in order to ensure the stability of the code and to aid in the development of the CDppEntity simulation. As we believe that the best testing is performed when extra code is not inserted into the deployment code, the unit testing was performed on the utilities.js file using QUnit as the framework for testing. In all, 271 assertions were made to functions of the utilities.js file to test data input parameters, ranges, and non-valid inputs. The test suite timing period was minimal and took of 84 ms for the completion. Integration or system testing was provided by Selenium IDE to valid the HTML web page interaction with the user. Chrome's addon SpeedTest was used to evaluate slow responsiveness periods for code correction. The interfacing between the RISE server and the WebLVC was performed by visual inspection.

In order to assess and document the CDppEntity Simulation page loading times and its dependencies on other services' response times, Firebug's Net profiler was exported and analysed. The page load times are not easy to define as it is dependent on the browser type, the speed of the JavaScript in the browser and the rendering times. However, loading time for each component is the time taken for each component - its latency - and is composed of the blocking, the Domain Name System (DNS) lookup, the connecting, the sending, the waiting and the receiving times. The reason for the assessment was to ensure that the page loading times were not significant to the "perceived loading times" which has a de facto standard of eight seconds and in part to examine the delay of the ESRI ARCGIS server to respond increased the time delays of the execution of the application until the loading of the map. Similarly, network throttling played an important part as a responsive and open bandwidth is demanded by the user for the changing of the scale of the map and the

downloading of the RISE resource results file. Loading of the various JavaScript files were examined to determine if the importing of the scripts had a detrimental effect on the user's perceived responsiveness of the webpage.

When the page loading times of a static instance are examined in Firefox, one can see that there is an apparent overall gain from caching on a high speed network as the loading times for cached and non-cached instances are almost similar to the user's "perceived loading time" as seen in Figure 41 and Figure 42. For Figure 41, with 99 requests in 1.99 seconds at the point where one starts the population of entities, 1.6 Mbytes were downloaded. Contrastingly, for the loading time for a cache for Figure 42, with 100 requests in 1.88 seconds, the point where one starts the population of entities, 1.6 Mbytes were downloaded in total with 606.1 Kbytes coming from cache. Significant in these figures is the fact that more downloads occur in the background and are required than realized through the direct requests of the CDppEntity Simulation.

A major challenge in testing the loading time performance directly is bandwidth as it can vary wildly. It is impossible to predict whether a site's performance is limited because of the network traffic or whether the server is loaded rather than the effects of the browser. Serious delays in the performance of the thin client in loading resources were significantly less than the user's "perceived loading time" delay. After this period, the user would clear the cache and reload the thin client in the hopes that the delay for the HTTP GETs would resolve itself for the initial load. In dealing with mash-up of services, this will always be a problem in that it doesn't meet the standards of repeatability.

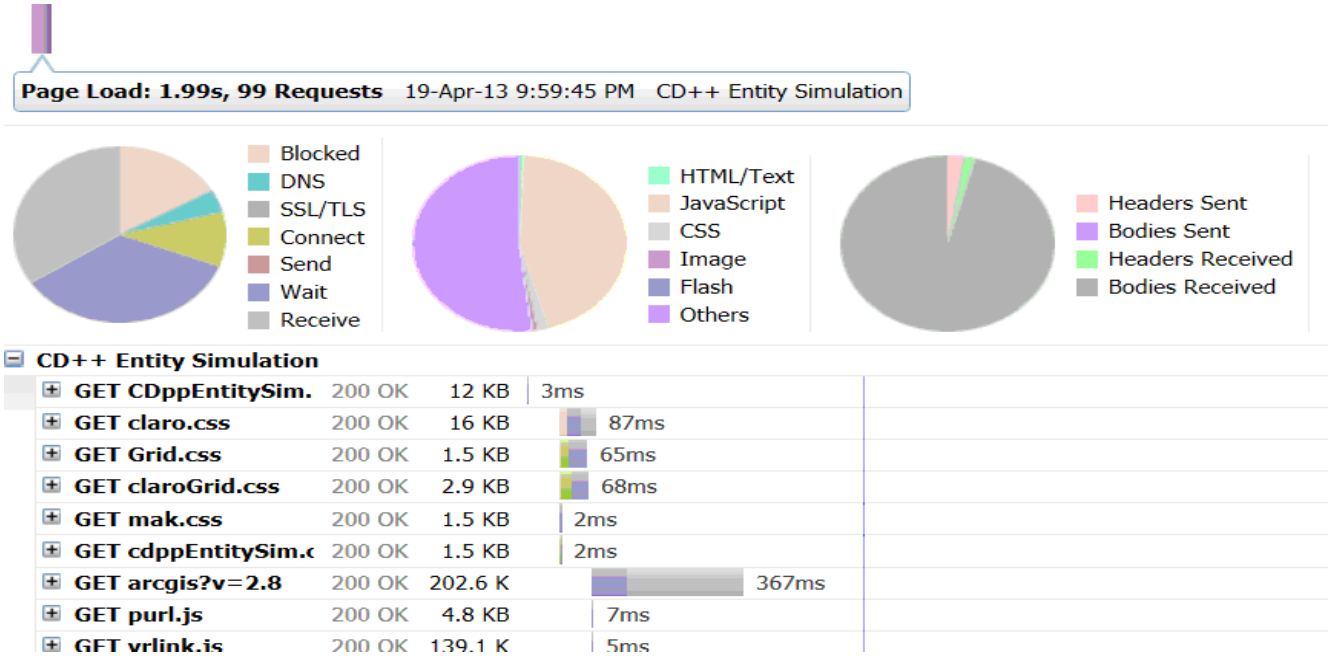

Figure 41. Application Loading Times Statistics - Uncached.

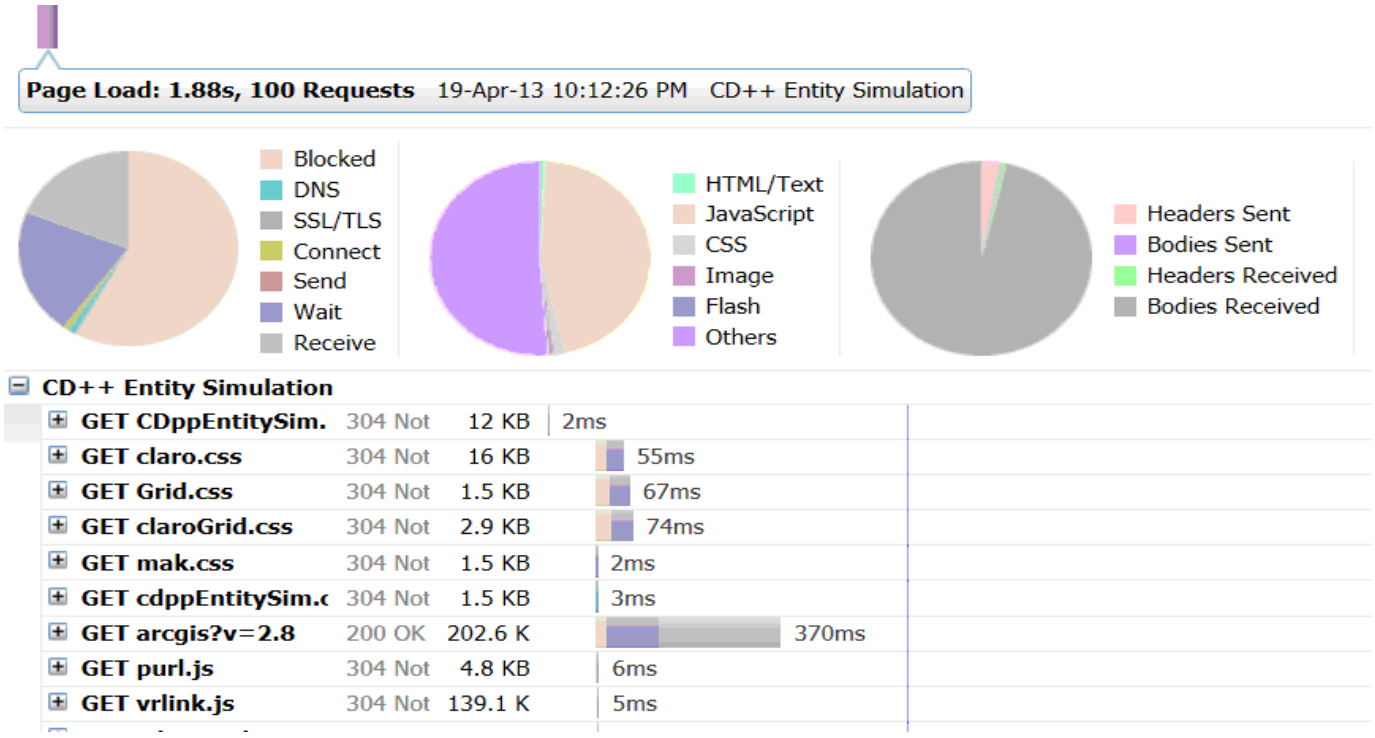

Figure 42. Application Loading Times Statistics – Cached.

Using Chrome's addon SpeedTest, Figure 43 summarizes the loading event times and the percentage of availability for CPU utilization and the times taken by the events for the main UI thread during the events of the browser loading period. Figure 44 illustrates the file

handling statistics. The 6.2% JavaScript Callback event is the time taken for the actual file handling, and decompressing and output as an ArrayBuffer while its internal thread communication for the bitjs library was insignificant at 20 milliseconds.

In order to evaluate the confidence interval for the timing responsiveness, [Ban10] was followed for a confidence interval of 100(1-α)%. In this case, replication was used and the replication sample is based on the ten models indicated earlier. Each model was executed ten times allowing a sample size of ten with the degrees of freedom equal to nine in order to determine the mean and confidence interval. The average time taken was the time taken for each function to extract and to ready the CDppEntity Simulation for populating its entities to the distributed simulation through the WebLVC server.

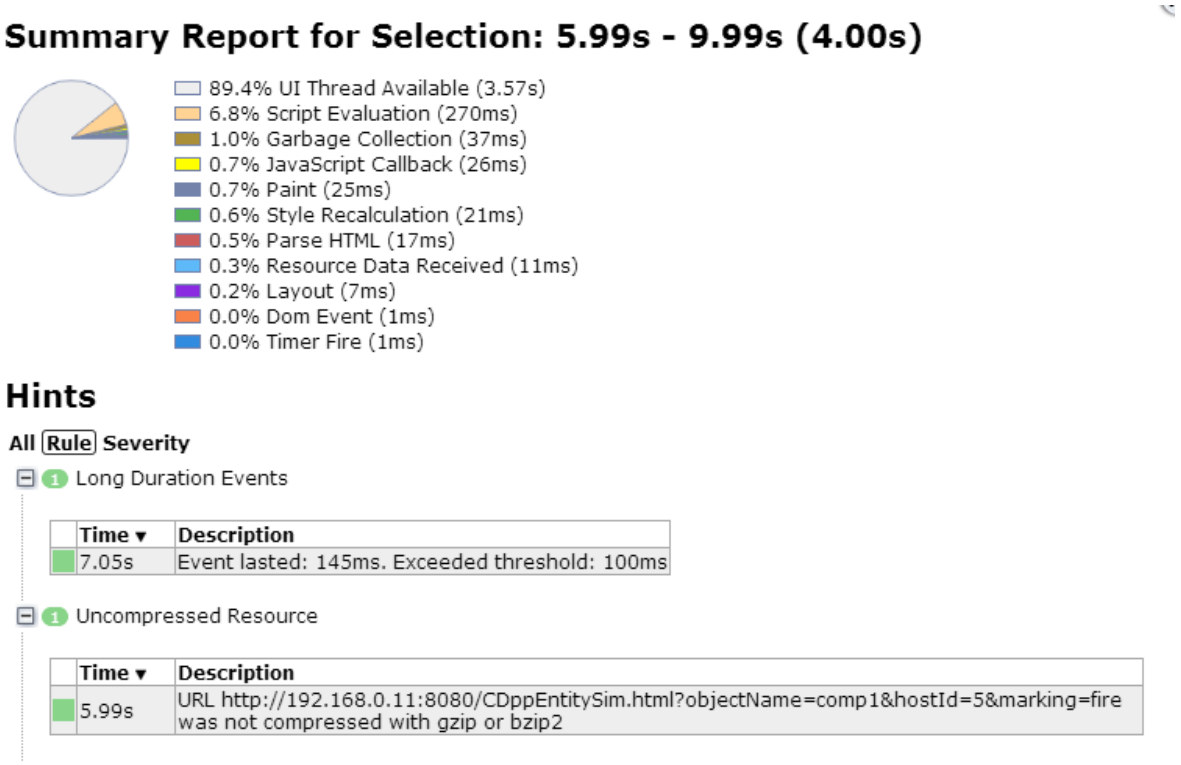

Figure 43. HTML Web Page Loading Statistics.

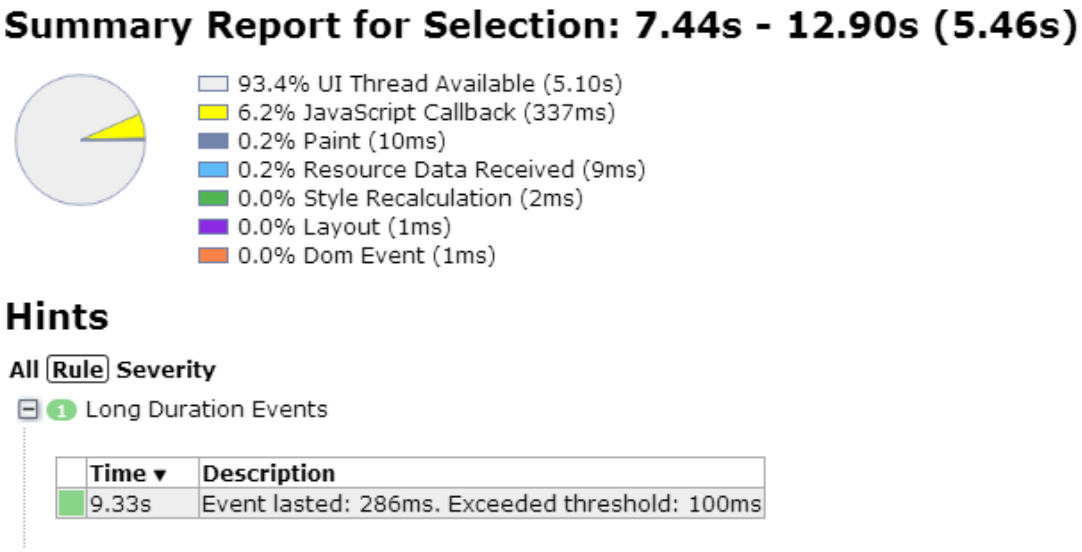

Figure 44. HTML File Handling Statistics.

Evaluating the function calls and object creation, the console.time function was used to evaluate the responsiveness to the CDppEntity Simulation. The fidelity of the console.time function was not accurate enough to measure microseconds and the values were reported with no significant digits. Thus, some of the times were hovering around 0 and 1 ms, which indicated that the function was processed but the times reported were inconsistently inaccurate for such minor time periods for the many trials. For the web page initialization routines after the loading of the map, the mean function and object creation times are given in Table 6.

Function	Mean (ms)	Std. Deviation	Confidence Interval (95%)
RestFulResource Object Creation	1	0	
Browser Type Checking Function	0	0	
Initializing Form Field Handlers Function	1	0	
CDppApplication Object Creation	1	0	
CDppApplication Initialization Function	1	0	

Dojo Base Map Creation Function	6.4	0.55	0.48
Initialization Function	21.8	0.84	0.73

Table 6. Initialization Function and Object Creation Times.

The main sequence routines from commencement to cessation to create the entities for population to the WebLVC server are sequentially detailed in Table 9 for the RESTful-CD++ DCD++ Fire Model. The largest delays in execution are caused by operating system calls in the file handling while the second largest delay occurred because of the creation of multiple graphic layers. This layer creation was performed during the operating system calls for an open dialog so that the user would not notice a delay in the layer creation.

Function	Mean (ms)	Std. Deviation	Confidence Interval (95%)
Multi-layer Creation Function	396.54	17.19	12.28
File Selection Function	402.60	21.29	15.22
RestfulResource initComms Function	19.00	1.25	0.89
Start Button Function	16.60	3.06	2.19
Zoom To Scale Function	3.40	1.17	0.84
Change State Function (start)	24.00	4.47	3.20
Entity Initialization Function	4.40	1.65	1.18
CDppEntity Initialization Function	1.00	0.47	0.34
Create Entity Graphic Function	2.20	0.92	0.66
Set Grid Data Function	7.00	1.25	0.89
Update Entity Graphics Function	23.90	5.09	3.64
Change State Function (End) (a and b)	25.90	8.98	6.42
a. Unload CDpp	25.30	8.46	6.05

b. Unload Application	0.60	0.52	0.37
Unload CDpp (composed of a and b)	45.00	14.17	10.12
a. Clear Stored Entities Function	23.20	7.21	5.15
b. Remove All Graphics Function	21.80	6.96	4.97

Table 7. Windowed Functions Operation Times.

different formats, to eliminate duplicate entries, and to generate positional information of each RESTful-CD++ grid cell coordinate relative to the origin latitude and longitude selected by the user. Also during this specified mean time the bearing, speed and line of advance speed of the entities from the origin are also dynamically generated. In comparing the values, and including the disk size of the files to the mean decompressing of the models in Table 8, a rough throughput can be determined for each format of the RESTful-CD++ resource results file. The figure is coarse because it includes disk access to get the web workers files, the creation of threads, their execution and its message posting.

RESTful-CD++ Resource File	Size (kb)	Mean (ms)	Std. Deviation	Confidence Interval 95%	Throughput (kb/sec)
DCD++ – Fire Model	220	373.40	54.47	33.76	589.18
DCD++ – Life Model	130	335.50	136.11	84.36	387.48
Lopez – FallRock Model	142	287.40	76.33	47.31	494.08
Lopez – Occupancy Model	912	1221.30	207.18	128.41	755.75

Table 8. Decompressing of a RESTful-CD++ Resource Results File.

Similarly for the data parsing shown in Table 9 of the decompressed log files, the throughput data parsing can be determined by dividing the uncompressed log file size by the average data parsing times of each model. Again this throughput includes the thread creation, their execution, and its message passing. Unfortunately, the two model format cannot be compared because of the non-Y output lines of simulation execution line messages in the DCD++ format, the log files generated for each RESTful-CD++ grid cell and

the multi-dimensionality of the two different formats. Presently, the DCD++ format contains two dimensional RESTful-CD++ grid cells (x, y) while the Lopez format contains three dimensional RESTful-CD++ grid cells (x, y, z).

RESTful-CD++ Resource File	Size (kB)	Mean (ms)	Std. Dev.	Confid Int 95%	Throughput (kB/sec)
DCD++ – Fire Model	3167	88.80	6.60	4.09	35664.41
DCD++ – Life Model	2305	68.90	6.74	4.18	33454.28
Lopez – FallRock Model	1157	156.70	7.56	4.68	7383.54
Lopez – Occupancy Model	13704	1563.90	17.23	10.68	8762.07

Table 9. Web Worker Data Parsing of RESTful-CD++ Log File.

Performance of the CDppEntity Simulation was dependent on the communication of the WebSockets with the WebLVC server. Using MÄk's Message Inspector, the entities could be examined and a packet rate given based on the number of messages given per second. However, the Message Inspector could not analyze the connections to the WebLVC server in order to examine the time and validity of the connections, the latency of the delivery of the messages, the message packet size, the reliability of the messages sent or the processing time of the messages [Pur13]. Presently, there are limited tools developed for this capability.

Using Chrome's network processing capability, the connection upgrade from TCP/IP to a WebSocket connection is shown in Figure 45. This packet is 47 bytes long and is variable as it exists as a JSON packet with the message kind identifier - 3 and the user specified client name identifier – CDpp Simulator. The frames of data sent to the WebLVC server as attribute message are approximately 448 bytes long depending on the entities object name.

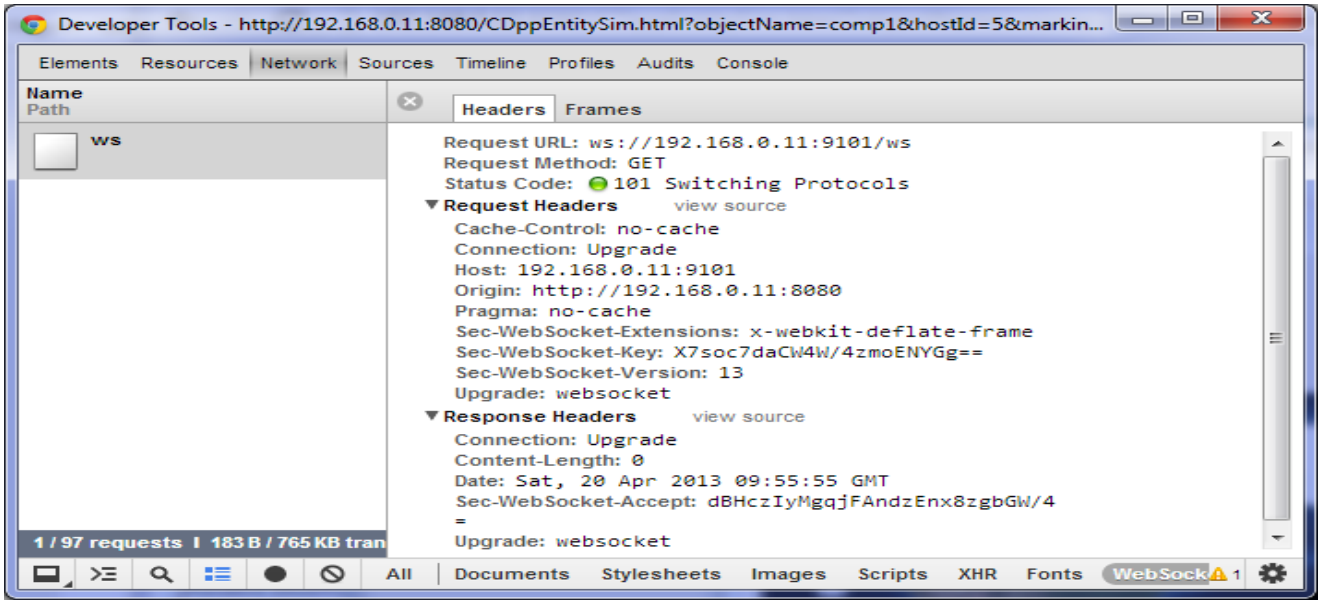

Figure 45. WebSocket Connection Details.

Figure 46 represents the WebLVC view of testing of three identical DCD++ fire models and a Lopez occupancy model hosted on different machines, spaced at different latitudes and longitudes, with different client application names, host identifiers and marking text.

Figure 46. WebLVC Server Entity Data.

Figure 47 represents the WebLVC view of the client applications' entity deletion messages being transmitted. Notice that the time period appears that the client side appears to send one message to delete all the entities; however, this is not true, but rather, this is due to the WebLVC's update rate as the client side deletes the entity data per entity.

Visual inspection using the supplied MÄk 2D viewer validates the CDppEntity Simulation operation in a web-based distributed simulation. North on Figure 48 is three DCD++ fires burning at different locations while a Lopez occupancy model is being generated south on Figure 48. The entities generated were on Firefox and Chrome browsers for the northern fires and the Internet Explorer for the occupancy model and 2D viewer.

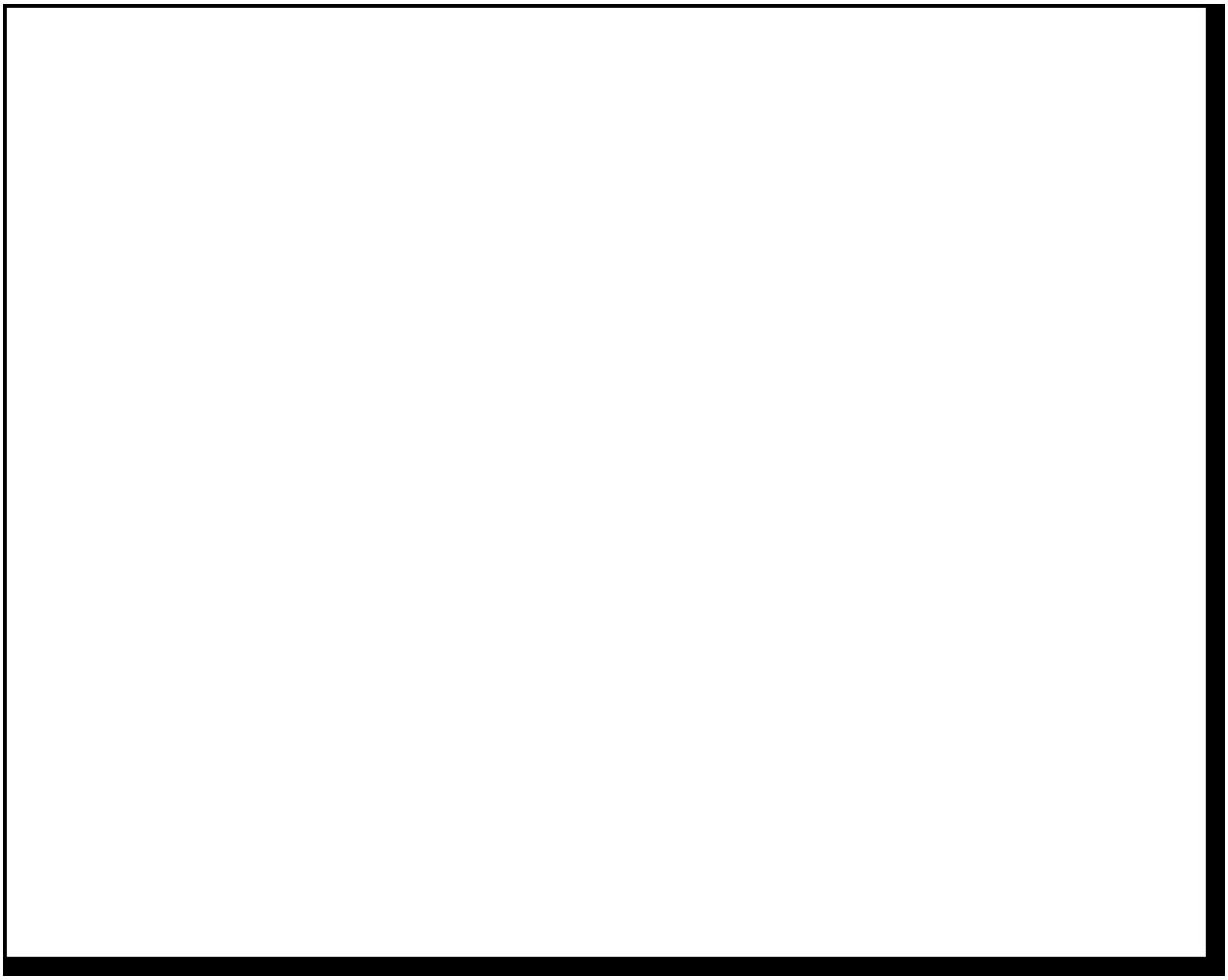

Figure 47. WebLVC Server Deletion of Entity Data.

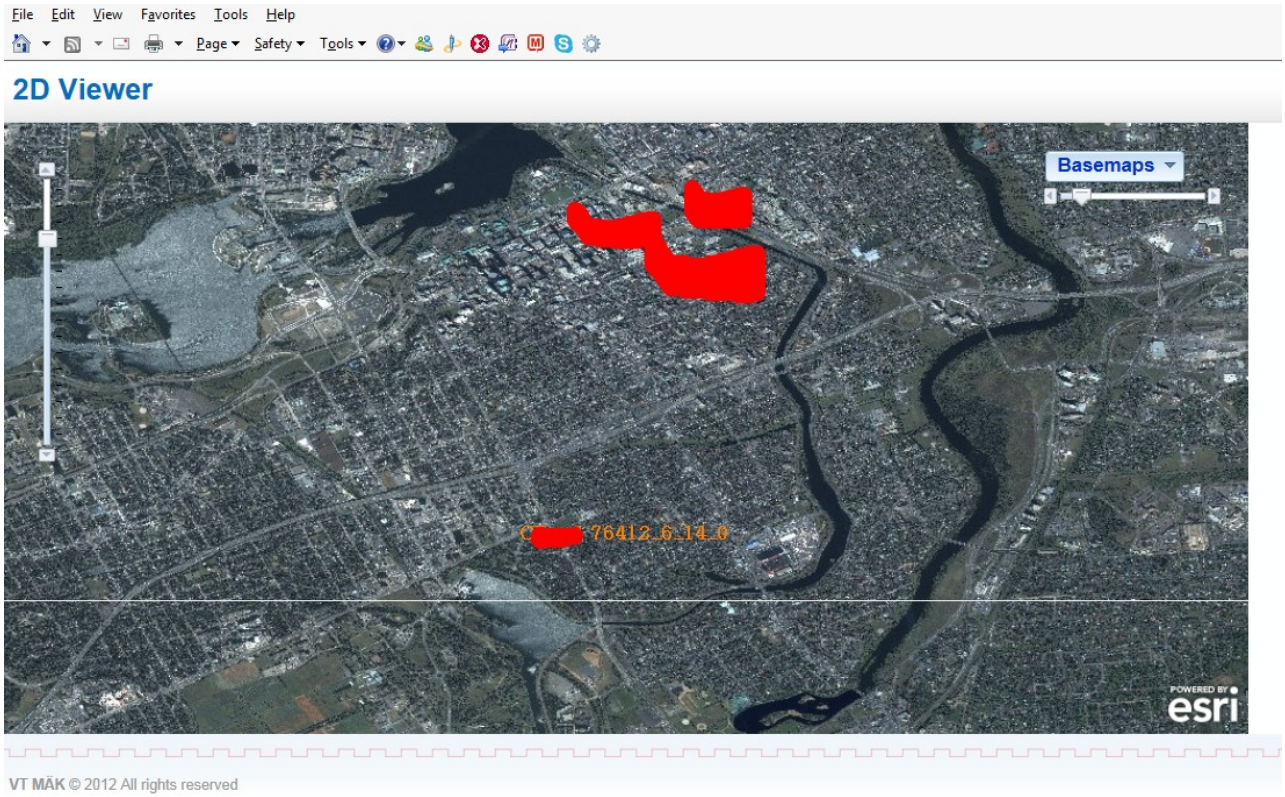

Figure 48. Visual Validation of the CDppEntity Simulation.

To corroborate the thin client technique during a distributed simulation and thus the RISE simulation results population via the thin client to a distributed simulation of which Figure 49 shows a one entity being logged internal to the distributed simulation. The standard value of 3000 was defaulted for a multicasting UDP port for a DIS environment. The WebLVC server was able to convert the JSON-based discrete event simulation into a DIS UDP packet and multicast it into the distributed simulation. Likewise, information contained in the distributed simulation was converted into the JSON-based data and displayed on the web-based thin client.

PDU Header		
	Protocol Version:	5
	Exercise ID:	1
	PDU Type:	1
	Protocol Family:	1
	Time Stamp:	Time Stamp: Relative : 1788000696
	PDU Length:	144
Entity State PDU		

Entity ID:		
	Site:	1
	Application:	1
	Entity:	22220
Force ID:		0
Number Of Articulation Params:		0
Entity Type:		4,1,0,0,0,0,0
Alternative Entity Type:		4,1,0,0,0,0,0
Linear Velocity:		X:0, Y:0. Z:0
Entity Location:		X: 1.10827e+006, Y: -4.34526e+006, Z: 4.52033e+006
Entity Orientation:		Psi: 1.81807, Theta: -0.778045, Phi: -3.13987
General Appearance:		
	Paint Scheme:	0
	Mobility Kill:	0
	Fire Power Kill:	0
	Damage:	0
	Smoke:	0
	Trailing Effect:	0
	Hatch State:	0
	Lights:	0
	Flaming Effect:	0
Space Platform Appearance:		
	Frozen Status:	0
	Power Plant:	0
	State:	0
Dead Reckoning Parameters:		
	Algorithm:	4
	Linear Acceleration:	X: 0, Y: 0, Z: 0
	Angular Velocity:	Psi: 0, Theta: 0, Phi: 0
Entity Marking:		
	Marking Char Set:	1
	Marking String:	Fire
Entity Capabilities		
	Ammunition Supply:	0
	Fuel Supply:	0
	Recovery Service:	0
	Repair Service:	0

Figure 49. DIS Distributed Simulation Entity Logging.

Chapter 6: Conclusion

6.1 Conclusions

In this dissertation, we have researched different technologies in order to determine how to create a client side interface that would expose discrete event simulation into a distributed simulation regardless of the FOM specifics. These technologies that were researched are as follows:

a. DCD++: how the DCD++ simulation engine operates and how it presents its simulation log results;
b. REST: what defines REST and how a REST server operates;
c. RISE: how the RISE operates, its internal structure, its RESTful statelessness, and access to its resources API based on URIs;
d. distributed simulations: how the distributed simulations are composed, the standards that they follow and the fidelity of their interactions;
e. DIS and HLA: the differences between the technologies, requirements and implementations;
f. WebLVC protocol: the proposed standard and partial implementation details of the WebLVC protocol to interface non-distributed simulations into distributed simulations;
g. MÄk simulation suite: how the simulation suite was developed, its implementation and its operation;
h. modern web technologies: HTML5 specifications, implementations, syntactic operation, processes, and extensions;
i. WebSockets: how WebSockets provide full duplex communications channels over a single TCP connection; and
j. multi-threading: how web workers are designed, their implementation, capabilities and limitations to enable multi-threading.

By combining the new web technologies of HTML5, WebSockets, multi-threading and the WebLVC protocol, a novel technique was developed that allowed the publishing of web

based simulation data into a distributed simulation regardless of the FOM specifics. This technique harnessed the power of the thin client to control the state of the RESTful discrete event simulation and automatically populate its simulation results into a distributed simulation. As a proof of concept, a thin client side application was developed to publish the complex simulation engine data into a distributed simulation environment available only over HTTP. The CDppEntity Simulation was able to access the resources of RISE, extract the pertinent log file, manipulate the data into a format acceptable to the WebLVC server, publish the data to the WebLVC server for web-based full-duplex communication of discrete event data into a distributed simulation. It also provided a visualization means of describing the time event simulation data.

The CDppEntity Simulation uses the new File API to access local file storage for a read-only capability and decompresses the RISE resource results file using web workers in an open-source library module called bitjs. The CDppEntity Simulation uses web workers to manipulate the Cell-DEVS grid cell coordinates to produce a bearing, speed, positional and distance calculation based on the Cell-DEVS origin. The manipulated data is formatted and incorporated into the data format required by the WebLVC server. The data is then transmitted via a WebSocket to the WebLVC server which transmits the data any connected distributed simulation.

6.1.1 Overcoming Challenges

The technical challenges that are a concern in the development of any technique were overcome as follows:

a. security:
    i. The results file was accessed from RISE as a compressed file containing different inclusions. By providing a combo box selection for the user, the thin client became knowledgeable about the extension type within the compressed file. Therefore a wrong setting on the combo box would indicate a compressed result file that was of the wrong type. In this case, the thin client would not select the correct file for data population and instead populate the data with superfluous data;

ii. Same-origin policy was handled the implementation of the user's administrative privileges. Instead of having the JavaScript directly making the requests to access other server's data, the user was required to interact during the initial page loading events. Without the user's interaction, the start command button was not made visible to commence the discrete event simulation population in to the distributed simulation;

iii. Obtaining the compressed simulation results file could not be accomplished without the user's interaction, as the XMLHttpRequest function, which is CORS compliant, was only designed for JSON, XML, HTML or plain text. Instead a iframe was populated with the discrete event simulation and its state was altered by the user;

b. simulation infrastructure: The simulation infrastructure was continued based on the MÄk prototype software. This required an understanding of JavaScript, Ajax, Dojo event handlers, and their respective limitations and capabilities. It also required an understanding of the developmental scope and purpose of the WebLVC server, and its capabilities and limitations. Though not itemized, these limitations were important as to the number of generated entities that could be created;

c. stability: In order to establish that the browser would not fail to operate when executed, measurement capabilities were developed to analyze the maximum number and size of elements that could be contained within the browser. The minimum value obtained (Chrome) was compared against the maximum size of all containers fully populated from the largest available simulation results file. It was determined that the files maximum size of 0.137 gigabytes was not near the failure point of the browser at 2.76 gigabytes. Also all functions were encased in try catch methods to prevent failure points of the thin client and provide exception handling capabilities. Message event calls and multi-threading used the inherent onerror calls to log error states; and

d. performance: The lifecycle of the entity generation and page loading functions were analyzed for performance and reworked until bottlenecks were removed. The timings developed for the functions were analyzed to determine the mean times of execution for ten models. Each model was run ten times to analyze its variability;

The developmental issues of a discrete event simulation had the following issues which are addressed as follows:

d. rendering complex visualizations to circumvent performance bottlenecks and threading issues: Checking that an object is contained in a container can be a performance issue. In order to prevent the thin client from performance issues, container examination for objects was performed in the background by the child threads. Each thread would respond with messages that would fire functional calls when valid data was present. This technique ensured that sequential operations were call only if valid data was present;
e. atomicity in data log operations to ensure consistency: The atomicity of data log operation was designed to handle the simulation execution result files. Failure to parse any component of the simulation results log file resulted in the non-population of entities by preventing the start command button from being displayed and
f. atomicity of data selection: The atomicity of data log selection was controlled by the combo selection box to ensure that either the correct format was selected or the thin client would return from execution with an warning modal dialog informing the user that execution could not continue because the set-up was incorrect. The idea was to provide an all-or-nothing capability.

6.1.2 Limitations and Constraints

Because of the possibility of high computational time of the simulation being executed on RISE for n-dimensional Cell-DEVS, the generation of dynamic, real-time simulation results were not evaluated. Training exercises are conducted in real-time with the ability to modify or stop parameters dependent on the evaluation or problems encountered. Thus, only completed simulation results were accessed and interfaced for simulation. Similarly, the software tools available on RISE limit the level of fidelity for visual representation by being an ASCII layered representation of the events.

Loading times of the CDppEntity Simulation were not reduced through the optimization such as reducing the number of sub-requests, streamlining or bundling CSS and JavaScript files, or HTTP compression. Though these are proposed to decrease the overall page loading times, it would be contrary to enabling or manipulating changes to individual modifications of the mash-up of services.

User agents commonly apply same origin restrictions to network requests. Client-side web applications are normally prevented from running from one origin and obtaining data retrieved from another origin such as RISE. This also thwarts unsafe HTTP requests that can automatically launch toward destinations that differ from the running application origin. As RISE is presently being refactored and there are different versions of RISE, we decided on applying the thin client technique so that it would remain independent of the hosting platform and remain capable of interacting as a mash-up of services across domains rather than altering RISE for CORS capability.

A major challenge in testing the loading time performance directly is bandwidth as it can vary wildly. It is impossible to predict whether a site's performance is limited because of the network traffic or whether the server is loaded rather than the effects of the browser. Serious delays in the performance of the thin client in loading resources were significantly less than the user's "perceived loading time" delay. After this period, the user would clear the cache and reload the thin client in the hopes that the delay for the HTTP GETs would resolve itself for the initial load. In dealing with mash-up of services, this will always be a problem in that it doesn't meet the problems of repeatability.

DIS, HLA, and TENA are different architectures and protocols. In HLA, the FOM is the vocabulary of the data that is interchanged within a federation and requires that the exposure of the data of a federate is defined by its SOM. Neither a FOM nor SOM has been created that is non-restrictive of discrete event simulation. Even though the hierarchical nature of the FOM can be extended to incorporate new federates, care must be maintained to ensure that names of objects, attributes, interactions and parameters remain constant. This naming restriction restricts the atomicity of discrete event models and forces the

discrete event modeller to understand the naming conventions of the FOM and alter his simulation results to the mandate of the FOM. The same is limitation is applicable to TENA. DIS, however, by using the enumerations standards, create generic wrapper for the different discrete event simulations. In the future once the WebLVC-to-HLA and HLA-to-WebLVC mapping are developed and standardized, the ability to distribute HLA non-FOM specific discrete event data can be realized.

WebSockets use encryption to ensure that validity of the WebSocket connection and to ensure the security of the connection. It does not provide a means to verify that the non-secure thin client is connecting to the correct sever. As such, it is assumed that the client knows the server and trusts the server's connection.

Limitations of the CDppEntity Simulation are dependent of the lifecycle development of the browser and their capability or refusal to implement all proposed features of web infrastructure, though the HTML5 standardization still slated for 2014.

Another limitation that can affect the CDppEntity Simulation is the firing of its timer cycle for the next RESTful-CD++ time events period. If the timer cycle period, which is based on the RESTful-CD++ time event periods, is too small, the client side application will not have enough of a period for execution and publication of entity data.

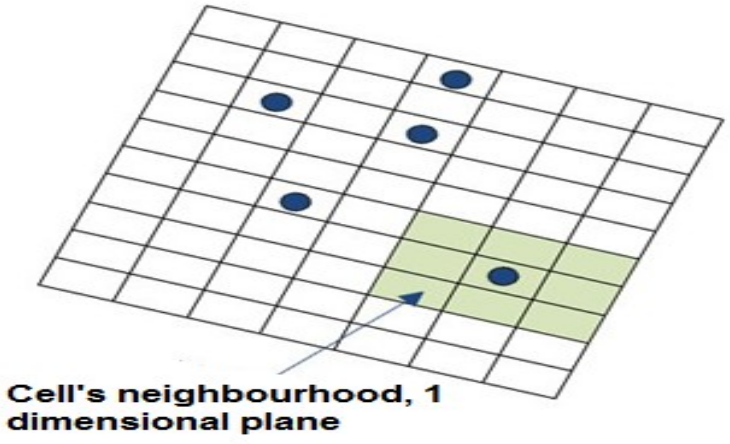

Figure 50. Cell-DEVS Neighbourhood Rotated by 20 Degrees.

A limitation to the entity generation is the need for linear rotation as the world map details real life positioning of physical structures while the generation of the RESTful-CD++ data is strictly in a heads up view shown in Figure 50.

During execution of the discrete event simulation, the entity type cannot be altered. The entity type is set during the file access function. Similarly, the location of the origin of the discrete event model is also set during the file access.

Finally, a limitation to the thin client side is its ability to parse the RESTful-CD++ data. In order to continue the use of the CDppEntity Simulation, future Cell-DEVS simulations should standardize the naming conventions and non-populated grid cell coordinate values use within the Cell-DEVS specification in CD++.

6.2 Summary of Contributions

In order to provide dynamic environmental entities as part of an agent-based modeling simulation, the paradigm of modeling required a complex modeling engine outside of the structure of the agent-based model yet interfaced and providing input into the distributed simulation. This dissertation contributed by:

a. discussing and examining of the different available architectures used in distributed simulations;
b. examining the problems of integrating half-duplexed transmitted web-based simulations into requirement-based monolithic applications of distributed simulations;
c. examining the coordination required to interface web based discrete event simulation into distributed simulations;
d. composing the discrete event simulation results such as the RESTful-CD++ simulation results into an interactive distributed simulation;

e. developing a proof of concept for the design and implementation of a thin client connected into a distributed simulation; and
f. examining and corroborating the WebLVC protocol in its operation as a bridge between web-based distributed simulations and monolithic distributed simulations.

The distributed simulation interface for the complex modeling environment has been solved by the creation of a mash-up of services called the CDppEntity Simulation. This thin client relies on the advances in technology of the web with HTML5, WebSockets and web workers to provide its mash-up of services for the RISE server and distributed simulations. Specifically, the developed thin client demonstrated the following behaviours:

a. exposed the simulation results of discrete event simulation, that is accessible through RISE through HTTP;
b. accessed and automatically manipulated and populated the simulation results into a format comprehensible to the WebLVC server for the distributed simulation;
c. published multiple distributed simulation entity data based on the discrete event simulation results file into a distributed simulation; and
d. provided a real-world visualization tool for the RESTful-CD++ simulation log results.

6.3 Future Research

Examination of the WebLVC protocol and the capabilities of the WebLVC server to translate web based simulation data into distributed simulations shows that not only web based discrete event simulations can be exposed for distributed simulations, but any other non-distributed simulation using the Web protocol can be exposed for a distributed simulation.

With regard to the CDppEntity Simulation, it is anticipated that future research could expend the effort of interfacing the CDppEntity Simulation directly into an HLA type distributed simulation. As the distributed simulations occur regularly with the Department of National Defence, it is possible to advance the distributed simulation into an actual battle training simulation. Modifications to the CDppEntity Simulation for a HLA federation would

require the modification of the CDppEntity class data into an acceptable data format for a FOM, and the development and standardization of the WebLVC protocol in its HLA-to-WebLVC and WebLVC-to-HLA mappings.

Modification for discrete event individuality of the RESTful-CD++ simulation data can be easily refactored in the RiseWebWorker parseArray function for the proper conditionals as each RESTful-CD++ simulation developer can define the input and output port number, names and specific values in the Cell-DEVS specification in CD++. Specifically, for example, the character positions in each data line can be altered for the population of integer or decimal positional values depending on the defined rule in the Cell-DEVS specification.

Another modification specific to the CDppEntity Simulation that could be implemented is the orientation rotation of the entities possibly through use of the 3D CSS. The 3D CSS is not presently standardized across all browsers, and its implementation is dependent on the browser. However, when standardized, the whole Cell-DEVS neighbourhood or each individual grid cell could be rotated from a heads up view to a situational view as shown in Figure 50 instead of applying specific individual model positional rules in the Moore neighbourhood defined through the Cell-DEVS specification.

Chapter 7: References

[AEG07] AEGIS Technologies Group. “IEEE 1516 HLA Hands-On Course”. AEGIS Technologies Group. 2007.

[AEG12] AEGIS Technologies Group. Pitch pRTI 1516 Support Webpage. Question and Answers. Available via http://www.pitch.se/support/pitch-prti-1516. [Accessed September 2012].

[Ahm05] Ahmed, M.; Yonis, K.; Elshafei, M.; Wainer, G. “Building a tool for modeling and simulation of computer networks”. Proceedings of the 38th IEEE/SCS Annual Simulation Symposium. San Diego, CA. U.S.A. 2005.

[Alz11] Al-Zoubi, K.; Wainer, G. "Using REST Web-Services Architecture for Distributed Simulation". 23rd Workshop on Principles of Advanced and Distributed Simulation. Lake Placid, New York, USA. 2009.

[Ame01] Ameghino, J.; Troccoli, A.; Wainer, G. “Models of complex physical systems using Cell-DEVS”. Proceedings of the 34th Annual Simulation Symposium. Seattle, WA. USA. 2001.

[Ban10] Banks, J.; Carson, J.; Nelson, B.; Nicol, D. *Discrete-Event System Simulation*. Prentice Hall. 2010.

[Ber96] Berners-Lee, T,. “Universal Resource Identifiers – Axioms of Web Architecture”. 1996. Available via http://www.w3.org/DesignIssues/Axioms.html. [Accessed October 2012].

[Bit13] “Binary Tools for JavaScript”, Available at https://code.google.com/p/bitjs. [Accessed February, 2013].

[Cro12] Crosbie, R., Zenor J., “High Level Architecture, Module 1 – Basic Concepts, Parts 1-6.” California State University, Chico. Available via http://www.ecst.csuchico.edu/~hla. [Accessed November, 2012].

[DEV08] Discrete Event System Specification Standardization Study Group, Discrete Event System Specification (DEVS) SG Final Report”. SISO-REF-019-2008. Available via http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=30802. [Accessed September 2012].

[DIS11] Distributed Interactive Simulation Product Support Group (SISO), “Enumerations for Simulation Interoperability”. SISO-REF-010-2011.1. Available via http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=32519. [Accessed October 2012].

[Dod99] “DoD Modeling and Simulation (M&S) Glossary” DoD 5000.59-M. Available via http://www.dtic.mil/whs/directives/corres/pdf/500059m.pdf. {Accessed September 2012].

[Fie00] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures.” PhD Dissertation. University of California, Irvine. U.S.A. Available via http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. [Accessed September, 2012].

[IEE12] “IEEE Standard for Distributed Interactive Simulation - Application Protocols,” IEEE Std 1278.1A-1998.

[IET11] Fette, I.; Melnikov, A., "The WebSocket Protocol". Internet Engineering Task Force. 2011. Available via http://tools.ietf.org/html/rfc6455. [Accessed December 2012].

[Kha05] Khan, A.; Wainer, G. "A visualization engine based on Maya for DEVS models". Proceedings of SISO Fall Interoperability Workshop. San Diego, CA. U.S.A. 2005.

[Lub12] Lubbers, P.; Greco, F. "HTML5 Web Sockets: A Quantum Leap in Scalability for the Web". Available via http://www.websocket.org/quantum.html. [Accessed December 2012].

[Mad05] Madhoun, R.; Wainer, G. “Modeling battlefield scenarios in Cell-DEVS”. Proceedings of SISO Fall Interoperability Workshop. San Diego, CA. U.S.A. 2005.

[MAK12] ‘WebLVC”. Available via http://www.mak.com/weblvc/. [Accessed October 2012].

[McF12] McFarlane, R., “An Introduction to the High Level Architecture Part 1”, McGill University, Montreal. Available via http://www.docstoc.com/docs/32971214/An-Introduction-to-the-High-Level-Architecture. [Accessed December, 2012].

[Mic11] Microsoft Security Research and Defense. “WebGL Considered Harmful”. Available via http://blogs.technet.com/b/srd/archive/2011/06/16/webgl-considered-harmful.aspx. [Accessed December 2012].

[Mol12] Möller, B.; Karlsson M. “A Management Overview of the HLA Evolved Web Service API”. Available via http://www.sisostds.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=27092&PortalId=0&TabId=105. [Accessed November 2012].

[Nat02] RTO NATO Modelling and Simulation Group Task Group 008. “NATO HLA Certification”. Available via http://ftp.rta.nato.int/public//PubFullText/RTO/TR/RTO-TR-050///TR-050-1to6.pdf. [Accessed December 2012].

[Pag98] Page, E.H.; Buss, A.; Fishwick P.A.; Healy K. J.; Nance R. E.; Paul R. J. “The modeling methodological impacts of web-based simulation”, Proceedings of the 1998 SCS International Conference on Web-based Modeling and Simulation. San Diego, CA, USA. 1998

[Pur13] Puranik, D; Feiock, D; Hill, J. “Real-Time Monitoring using Ajax and WebSockets”. Available via http://cs.iupui.edu/~hillj/PDF/ecbs2013-websockets.pdf. [Accessed February 2013].

[Ray13] “Ray Tracing”. Available via <nerget.com/rayjs-mt/rayjs.html>. [Accessed Mar 2013].

[SBES06] National Science Foundation Blue Ribbon Panel on Simulation-Based Engineering Science. SBES: SBES: Revolutionizing Engineering Science through Simulation. Technical Report. Available via: http://www.nsf.gov/pubs/reports/sbes_final_report.pdf. [Accessed November, 2012].

[SISO07] Distributed Interactive Simulation Product Development Group (SISO), “Guide for: DIS Plain and Simple”. SISO-REF-020-2007. Available via http://www.sisostds.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=29302&PortalId=0&TabId=105. [Accessed October 2012].

[Smi95] Smith, K. “Distributed Interactive Simulation Network Manager”. ARL-TR-780. Army Research Laboratory. USA. 1995. Available via http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA298053. [Accessed December 2012].

[Str01] Straßburger, S. “Distributed Simulation Based on the High Level Architecture in Civilian Application Domains.” PhD Dissertation. Otto-Von-Guericke University, Magdeburg. Germany. 2001.

[Tro03] Troccoli, A., Wainer, G. "Implementing Parallel Cell-DEVS". Proceedings of 36th IEEE/SCS Annual Simulation Symposium. Orlando, FL. USA. 2003.

[W3C12 - FA] “File API” W3C Working Draft 25 October 2012. Available via http://www.w3.org/TR/FileAPI. Accessed November 2012. [Accessed December 2012].

[W3C12 – HTLM5] "HTML5 Differences from HTML4" W3C Working Draft 25 October 2012. Available via http://www.w3.org/TR/html5-diff/. [Accessed December 2012].

[W3C12 - WW] "Web Workers" W3C Candidate Recommendation 01 May 2012. Available via http://www.w3.org/TR/workers/#sharedworker. [Accessed December 2012].

[W3C12 - CORS] "Cross-Origin Resource Sharing" W3C Working Draft 3 April 2012. Available via http://www.w3.org/TR/cors. Accessed December 2012.

[Wai00] Wainer, G. "Improved Cellular Models with Parallel Cell-DEVS". *Transactions of the Society for Computer Simulation International.* Vol. 17(2), pp. 73-88. June, 2000.

[Wai01] Wainer, G.; Giambiasi, N. "Timed Cell-DEVS: modelling and simulation of cell spaces". Invited paper for the book Discrete Event Modeling & Simulation: Enabling Future Technologies. Springer-Verlag. 2001.

[Wai02] Wainer, G. "CD++: a toolkit to develop DEVS models". *Software - Practice and Experience.* vol. 32, pp. 1261-1306. 2002.

[Wai04] Wainer, G. "Modeling and Simulation of Complex Systems with Cell-DEVS". Proceedings of the 2004 Winter Simulation Conference, Washington DC. 2004.

[Wan12] Wang, S. "Restful Interoperability Simulation Environment (RISE) Middleware". 2012.

[Wea11] Wessels, A.; Purvis, M.; Jackson, J.; Rahman, S."Remote Data Visualization through WebSockets". 2011 Eighth International Conference on Information Technology: New Generations (ITNG), pp. 1050-1051.

[Wil94] Wilson, A.; Weatherly, R. "The Aggregate Level Simulation Protocol: An Evolving System". Proceedings of the 1994 Simulation Conference: pp. 781-787.

[Zap11] M. Zapatero, G. Wainer, R. Castro, M. Houssein, "Architecture for integrated modeling, simulation and visualization of environmental systems using GIS and Cell-DEVS" in Simulation Conference (WSC), Proceedings of the 2011 Winter, 2011, pp. 997-1009.

[Zei98] Zeigler, B.; Ball, G.; Cho, H,; Lee, J.; Sarjouhian, H. “The DEVS/HLA Distributed Simulation Environment And Its Support for Predictive Filtering”. Available via http://www.cs.mcgill.ca/~hv/articles/Quantization/UnivArizonaCDRL2.pdf. [Accessed January 2013].

[Zei99] Zeigler, B.; Hall, S.; Sarjouhian, H. “Exploiting HLA and DEVS To Promote Interoperability and Reuse in Lockheed’s Corporate Environment”. 1999. Available via http://acims.asu.edu/wp-content/uploads/2012/02/Exploiting-HLA-and-DEVS-To-Promote-Interoperability-and-Reuse-in-Lockheeds-Corporate-Environment.pdf. [Accessed January 2013].

[Zei00] Zeigler, B.; Kim, T.; Praehofer, H. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press. 2000.