RESTful Collaborative Modeling

by

Jerome B. Heiser

An Applied Project Presented in Partial Fulfillment

of the Requirements for the Degree

Master of Engineering

Approved May 2013 by the

Graduate Supervisory Committee:

Hessam Sarjoughian, Chair

ARIZONIA STATE UNIVERSITY

May 2013

ABSTRACT

This capstone project examines a collaborative modeling approach using RESTful methodology applied across a disparate simulation environment. With the advent of online education, new opportunities exist for developing methods for collaborative modeling at a distance. These opportunities exist at the internet level (Universities, Company Divisions, and International Research) and at the intranet level (Company Complex, University Campus). Coupled with the fact that modeling exercises are becoming increasingly complex for one individual to master, system modeling demands collaborating amongst subject matter experts usually geographically disbursed. Toward that end, this project focused on several design concepts that resulted in the composition of models through a web-based interface based on a mater-slave paradigm. The project design concepts exercised included the ASP.NET Model-View-Controller (MVC) framework in Visual Studio 2012, the JNBridge software to traverse C# to Java code, and the DEVS-Suite Simulation Environment to develop the RESTful Collaborative Modeling System (RCMS) middleware. Model designs were chosen from the CSE561 Modeling & Simulation course project titled “Modeling Interdiction Methods” [16], to integrate into a controlled intranet environment that represented the RCMS. Although a fully integration and tested collaborative model was not realized, the projects objectives (a developed RESTful architecture, an established interface supporting interactive collaborative modeling, and a developed user interface to support configuration, status and control) were satisfied. Security concerns were addressed throughout the design and development process, however complete security extensions to the systems were not incorporated and were identified for future work in this area.

ACKNOWLEDGEMENTS

First I would like to express the deepest appreciation to the committee chair and advisor, Dr. Hessam Sarjoughian for his steadfast support and guidance. I would also like to thank Wayne Citron and Paul Summers from JNBridge for their help with integrating and understanding the JNBridgePro product and capabilities. Special thanks goes out to artists Boney James, Rick Braun, Paul Hardcastle, Peter White, Richard Elliott, Dave Koz, Bernie Williams, Kim Waters, Najee, Brian Culbertson, Bob Baldwin, Marc Antoine, Brian Simpson, Warren Hill, Michael Lington, Lee Ritenour, Sting, David Benoit and Nelson Rangell for their smooth jazz that carried me through the long nights working on three computers in my study. Finally, I would like to thank my wife Debbie for her understanding and support in the many hours devoted to this project.

Table of Contents

List of Figures

List of Tables

OBJECTIVE

This capstone project examines a collaborative modeling approach using a RESTful methodology that can be applied across disparate simulation environments. The project focuses on three objectives: (1) Developing a RESTful Collaborative Model System (RCMS) modeling approach, (2) developing model-component/REST-applications hosted as a resource, and (3) the developing model interaction through a Model-View-Controller (MVC) framework.

Developing the RCMS approach involved looking at the RESTful methodology [1] in the interaction of the simulation model as a web-based service. The objective is to use HTTP POSTs and GETs to form the basis of communication for the modeler and the model in this approach. Application development with a master-slave mentality evaluates definition of model coupling and interaction.

Component/application remote hosting as unique models looks at the interface protocol to support the collaborative modeling approach. Application Program Interface (API) to extend DEVS-Suite to exercise models as applications in the RCMS approach is investigated. The project evaluation utilized model designs from the Modeling Interdiction Methods course project [16] design concept in a controlled intranet environment to integrate with RCMS.

The project's user interaction focuses on the functionality provided by the ASP.NET Model-View-Controller (MVC) framework in Visual Studio 2012 in developing the RESTful Collaborative Modeling System (RCMS) middleware. Restful Collaborative Model (RCM) and Restful Collaborative Node (RCN) views are investigated as the basis for user interaction in the collaborative modeling exercise and to support model features and modeler dialog communication.

MOTIVATION

With the advent of online education becoming more popular, new opportunities exist for developing methods for collaboration in project and research endeavors as well as training exercises orchestrated at a distance. These opportunities exist not only at the internet level (Universities, Company Divisions, and International Research) but also at the intranet level (Company Complex, University Campus). Modeling exercises are becoming increasingly complex and one individual cannot hope to master the understanding of all systems, and therefore must rely on subject matter experts to meet the demands in an efficient manner for project or research efforts. A number of investigations in the area of distributed multi-model development using Service Oriented Architecture (SOA) methods with the view toward collaboration have been conducted [2][3][4]. They seek to address the interoperability concerns associated with the nature of working collaboratively over the internet, using disparate platforms and different simulation environments. This project continues in that direction but with a move toward a more simplified interface using Representational State Transform (REST) communication and exercising the full value of resource access in that process.

This project seeks to address the view of modeling, rather than execution, and the concepts involving interoperability and communication between stakeholders collaborating in the modeling exercise. The RESTful Collaborative Modeling System (RCMS) methodology approaches the pragmatic, semantic, and syntactic levels of interoperability through user interfaces designed to allow open and flexible communication. The pragmatic level of the receiver's interpretation of message content in terms of the sender's intent, the semantic level of attribute definition as it relates to each model component, and the syntactic level focusing on the communication structure exposed through definition are the aspects investigated in this project.

The RESTful approach investigated was motivated by the concept of the RESTful CD++ methodology developed by Al-Zoubi and Wainer [5] resulting in middleware to support a distributed simulation with disparate platforms and environments. The concepts of resource utilization as the prime communication mechanism to exact events, data flow, and control of the simulation exercise following from this approach, is the methodology examined in this project.

Coupled with the RESTful approach is the adoption of the Model-View-Controller (MVC) paradigm to support the RCMS integrated approach with the simulation environment and the user. Learning and using Microsoft's ASP.NET MVC Framework serves as the motivation to apply the MVC approach.

BACKGROUND

This project design concepts for RCMS focus on four main areas; REST, MVC Framework, CD++, and DEVS-Suite. Each of these areas is addressed in the following sections to provide an introduction into the design discussion to follow.

Representational State Transform (REST):

The REST architecture was first presented in the doctoral thesis of Roy Fielding [6]. In his thesis he refers to REST as an architectural style[1] to be used as a guide in the design and development of software behavior for the Web. This behavior is associated with clients and servers, and exemplifies a set of five software engineering principles which he referred to as the Representational State Transfer (REST). They consist of the following [7]:

- Addressable resources
- A uniform constrained interface
- Representation-oriented
- Communicate statelessly
- Hypermedia As The Engine of Application State (HATEOAS)

The five principles that drive the REST architectural style can best be described as consisting of a set of constraints applied to the system design at hand. When used in conjunction with the Hypertext Transfer Protocol (HTTP) protocols it is referred to as being RESTful[2]. In that context those principles can be defined as follows:

Addressable Resources: Identify data and information in your system as a resource and give it an ID. Make that ID addressable and unique through the use of the Universal Resource Identifier (URI). The

1 An architectural style is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style.[5]

2 Note that REST is not limited to the HTTP protocol provided those application protocols already support uniform "verbs" based on the transfer of representational state.[6]

idea here is that your resource then becomes reusable and linkable, thereby allowing you to pass links from one resource to another.

Note that the format of the Universal Resource Identifier (URI) is standardized, and a good, compact definition with examples can be found in Burke's book on REST [7] and Scott Allen's book on HTTP [14].

Uniform Constrained Interface: Apply a uniform set of well-defined methods to the resources in your system. For REST they consist of the well-defined HTTP protocol methods of PUT, GET, POST, DELETE, HEAD, and OPTIONS. What this means is that you do not have any parameters or verbs within the service request that "define the action" you want to perform; you use only the HTTP methods.

Representation-Oriented: Interact with resources through the format representative of the resource platform. Through HTTP you have the concept of an accept header which you can request the format for representation of the communication. Different platforms need different formats like browsers which need HTML, or JavaScript which needs JavaScript Object Notation (JSON). Extensible Markup Language (XML) is the most preferred format and is supported by all platforms. Representations are communicated between client and server through the HTTP methods with specific URIs. This allows receiving the current state of the resource with a GET, or passing a resource representation with a POST or PUT so that its state can change in the server resource, if allowed. The interactions are in the representation; each communication, that is each message request or response, provides enough information to describe the processing necessary for that message.

Communicate Statelessly: Do not maintain client state information in a session. Rather develop in terms of a resource state. So instead of maintaining the information regarding the state of the session, that is storing the client state on the server, use the link to the resource and let the server manage the state of the resource; the client maintains the links. For example, if I were a customer shopping on the internet and I have a shopping cart instead of maintaining the cart information as the

state, refer to the cart as a resource and link to it. The cart as a resource is then the state of the client shopping.

Hypermedia[3] As The Engine Of Application State: Use the resource IDs to link objects together. The transitions that you make within your application can use the representational requests and response formats (header identifier Referer) to drive you to other resources. You can also aggregate things from multiple sources by using attributes that contain the links to those resources within your communication. So for example executing a purchase on the internet, the response to your request can contain the link to the next interaction resource as well as the functions available to transition to that state.

Why REST? The question can be addressed as follows: (1) REST is highly scalable in that there is no constraint on how resources are referenced; there is no implicit binding or service contract on actions allowing the architecture to scale to the needs; (2) REST supports generality in that by having general paradigms and protocols anybody can interact with the interface regardless of underlying architectures and software; (3) REST allows independence in that you can deploy features and components independent of others using the REST paradigm; (4) REST supports caching to deal with latency; security is built into the headers as defined by the Http specification; and (5) REST supports encapsulation in that data is abstracted from the application functionality [8].

Model View Controller (MVC)[4] Framework:

The term model-view-controller has been in use since the 70's and refers to a software pattern for application development dealing with Graphical User Interfaces (GUI) applications. The MVC pattern refers to an application that consists of three pieces: a ***model*** which typically represents a domain model

[3] Hypermedia is a computer-based information retrieval system that enables a user to gain or provide access to texts, audio and video recordings, photographs and computer graphics related to a particular subject. Hypermedia is a term created by Ted Nelson. **Hypermedia** is used as a logical extension of the term hypertext in which graphics, audio, video, plain text and hyperlinks intertwine to create a generally non-linear medium of information.[6

[4] The context for this overview was taken from reference [9].

which contains data in a business domain as well as operations, transformations, and rules for manipulating that data; a *view* which renders some part of the model as a user interface; and a *controller* which processes incoming requests, performs operations on the model and selects views to render to the user [9].

The *model* can also be viewed by what it is not responsible for; it does not render data and it does not process requests. The model represents the domain or the activity that the application is to perform, and therefore is referred to as the *domain model.* The domain model then contains domain types which are the classes and structures of the model where the operations are the methods defined to operate on those domain types. The domain model is persistent and usually handled in concert with a relational database, i.e. that while persistence is viewed as part of the model, the data persisted is kept in an independent repository that interacts with the database; the model only interacts with the repository.

The *view* contains the logic to display information to the user, and that only. The *controller* bridges the model and the view; it processes the incoming request and decides what view to present and what operation the model is to perform.

The MVC pattern then represents the separation of concerns where the logic to manipulate data is contained only in the model, the logic that displays the data is only in the view, and logic that handles the user requests and input is only in the controller. The interaction in the MVC pattern can be represented pictorially as shown in Figure 1. It represents the manner in which REST methodology meshes with MVC.

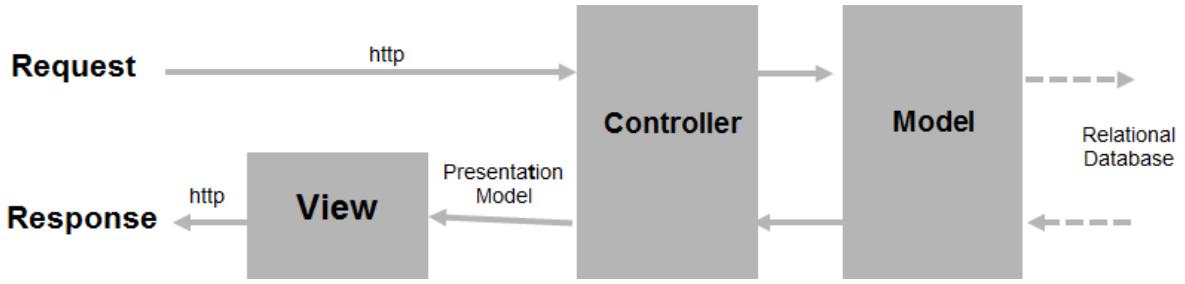

Figure 1 – MVC Application Interactions

The MVC Framework in Visual Studio 2012 handles these concerns seamlessly through the Entity Framework (EF) using the Code First approach. It introduces the Object Relational Mapping (ORM) in the .NET environment. Conceptual model classes (coded first) are mapped to database schema (automatically), generating the tables in the database. The objects generated through queries are managed to support persistence with the database and interact seamlessly within the views and controllers. Entity Framework also introduced support to track changes to Plain Old CLR Object (POCO) class' and to the database in the MVC Framework. The migration tools provide ease of update and change management control throughout the development process [22][23].

RESTful CD++ Middleware:

RESTful CD++ represents distributed simulation middleware that exposes services through URIs [1][10]. It routes the received request to the appropriate destination resource and applies the required Http method on that resource (GET, PUT, POST, UPDATE, and DELETE). The REST CD++ methodology utilizes a URI-oriented architecture in a hierarchical fashion (parent-child structure) to segregate users and user-services. The middleware is meant to remain independent of the service represented in the simulation engine.

The URI template depicted in Figure 2 reflects the RESTful Interoperability Simulation Environment (RISE) and Application Program Interface (API) design [12][13], formerly known as REST CD++; it is the template followed in this project. The hierarchy within this template adheres to a common structure denoted as "…<{userworkspace}/{servicetype}/{framework}>". In this structure, {userworkspace} refers to a unique workspace to avoid naming conflicts; {servicetype} refers to a unique simulation service or component to allow multiple services to exist in the hierarchy as is the case in this project; and {framework} which refers to the unique experiment or modeling exercise being conducted.

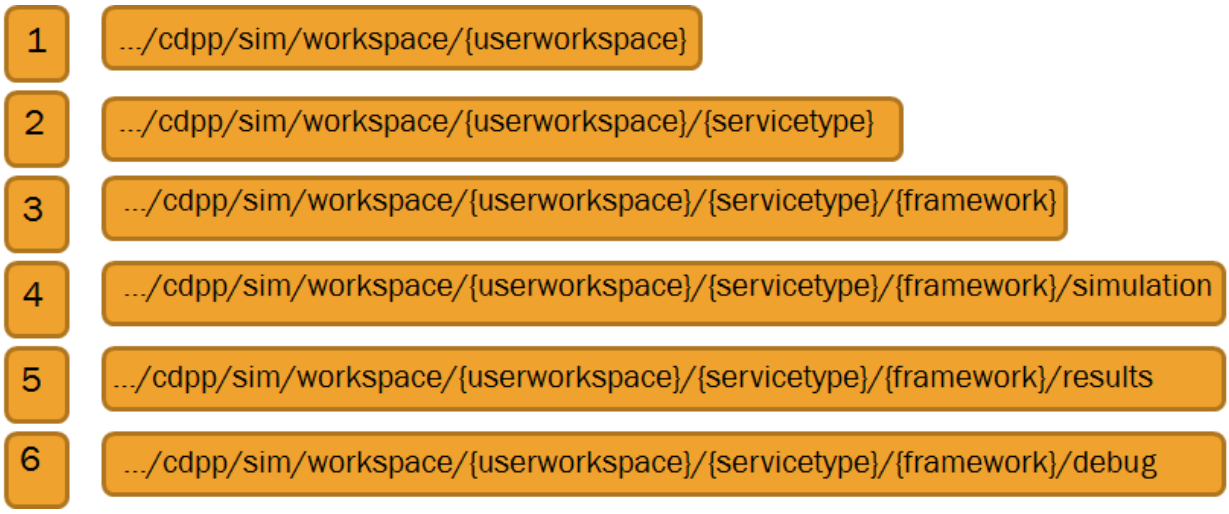

Figure 2 - RISE URI Template

The context associated with the RISE design will not be employed in this project. But the concept of URI hierarchy, database persistence of URIs, and basic template structure will be followed in implementing the design methods for the project as outlined in the Design Discussion section to follow.

Discrete Event System Specification (DEVS) - Suite:

Among discrete event modeling approaches, the Discrete Event Systems Specification (DEVS) is well suited for formally describing concurrent processing and the even-driven nature of arbitrary configuration of network systems. This modeling approach supports hierarchical modular model construction, distributed execution, and characterizes complex, large-scale systems with atomic and coupled models. DEVS-Suite is an open source, general-purpose discrete event simulation environment that supports DEVS model execution and evaluation.

Within the DEVS-Suite environment, hierarchical models can be displayed as components with support for animation of message flow amongst coupled atomic models as well as the atomic model states. DEVS-Suite supports the complex task of simulation modeling, offering on –the-fly configuration of experiments through a GUI and visualization of component behavior as time-based trajectories as depicted in Figure 3. The DEVS-Suite consists of the DEVSJAVA simulator kernel, the SIMVIEW for hierarchical viewing of models and their message-passing animation, TIMEVIEW for plotting time trajectories, DEVS Tracking Environment for automated design of experiments, and controller for time-stepped execution.

In DEVS-Suite, execution of the models can be tracked as both the animation of input/output messages for the atomic/coupled models, and the state changes of the atomic models, as well as log files of the results. Simulation experiments can be triggered with test inputs which can be selected

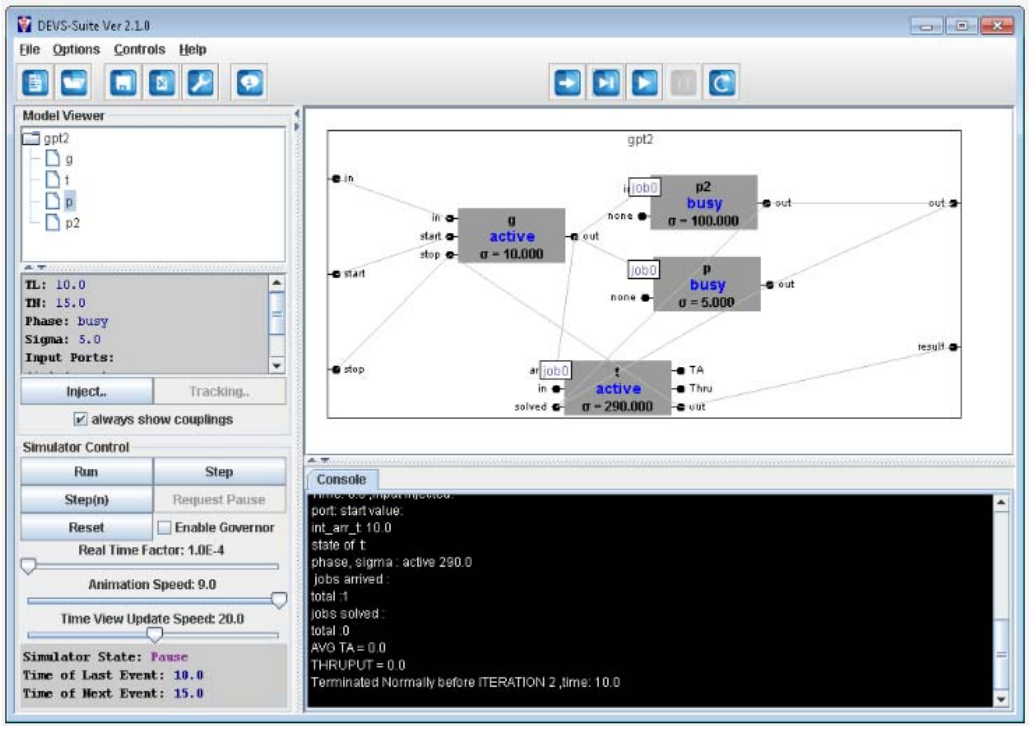

Figure 3 - DEVS Suite Simulator User Interface[5]

through a dialog to start the model execution. At the completion of the simulation run, statistical outputs and trajectories can be displayed for pre-defined phase and sigma state variables.

Figure 4 shows the systematic representation of the basic atomic model that supports the DEVS formalism. The model represents a set of input ports through which external events are received, a set of output ports through which external events are sent, a set of state variables and parameters (includes "sigma" and "phase" where in the absence of external events the model stays in current "phase" for time given by "sigma"), a time advance function which controls the timing of internal transitions, an internal transition function which specifies to which state the model will transfer after the time advance has elapsed, an external transition function which specifies how the model changes state when an input

[5] This figure was obtained from a power point presentation listed as reference [15].

is received, a confluent transition function which is applied when an input is received at the same time that internal transition is to occur, and an output function which generates an external output just before an internal transition takes place [17].

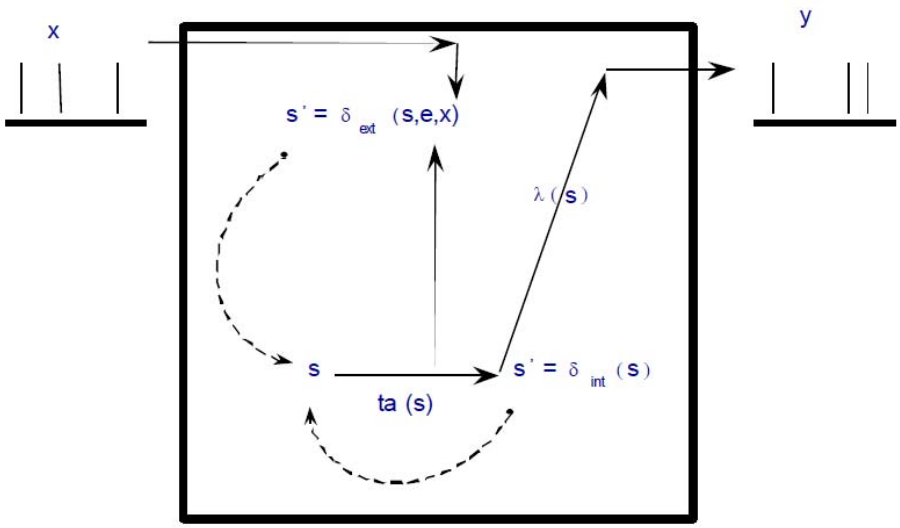

Figure 4 - Systematic Representation of an Atomic Model[6]

The DEVS formalism has been represented in DEVS/C++ and DEVSJAVA [27], and extended through DEVS/HLA [28], DEVS/CORBA [29], cell-DEVS [30] and DEVS/RMI [31]. The DEVS Modeling Language (DEVSML), built on XML, also provides model interoperability at remote locations although not as a distributed simulation approach [25]. Also note that the JavaWebStart[7] supports complete execution of standalone Java applications from a Web server.

[6] Drawing is a screen shot taken from reference [17]
[7] http://acims1.eas.asu.edu/WebStarts

DESIGN DISCUSSION

The distributed simulation approach for this project was developed to execute across multiple platforms with the view of each platform performing the simulated functions of one node of a complex system, allowing for collaboration on an overall system model. For example, the system concept CENTRA^2 [20] shown in Figure 5 depicts several complex systems, each of which can be modeled on a separate platform by a subject matter expert (SME) modeler cognizant in that area of design. In the simulation approach presented in this project, the modeler's simulation environment can be agnostic to the overall model architecture allowing flexibility to the modeling approach, removing simulation environment constraints and providing a more efficient method to reach the end goal or objective of the effort.

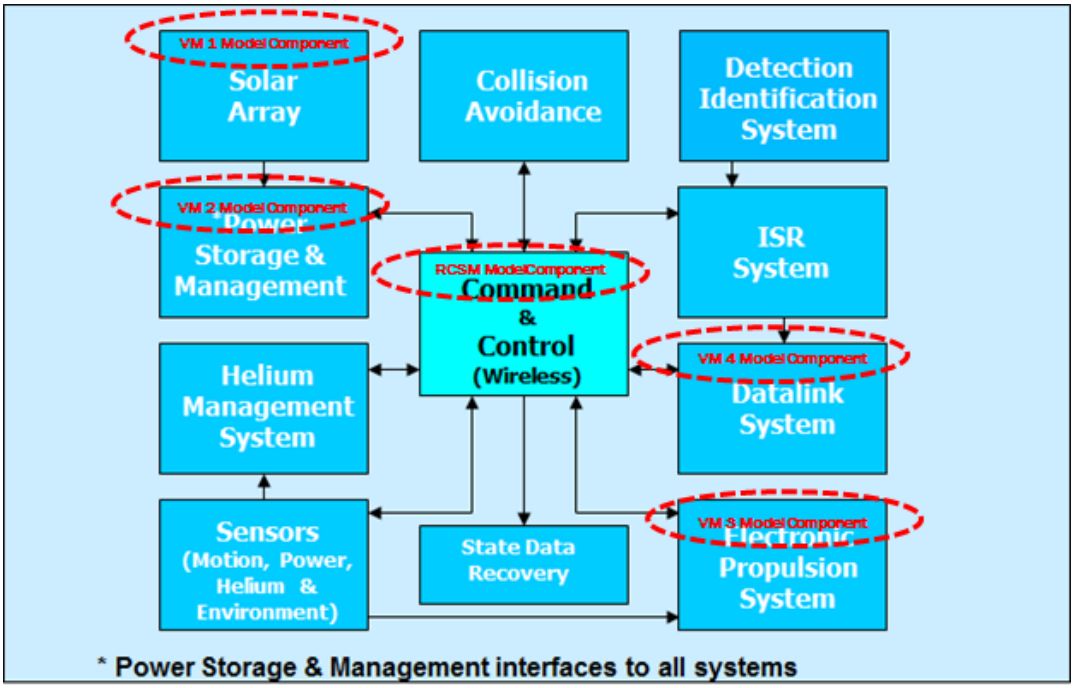

Figure 5 – High Level CENTRA^2 Model Architecture

This approach also allows the model to be developed in a progressive manner as suggested in Figure 5, adding components in an incremental fashion through the development cycle. The initial effort can be associated only with those nodes or components of the overall model design indicated by the circled-block-identifiers, allowing the system to be developed in future phases to meet such constraints as cost profiles, equipment availability, or SME (modeler) availability.

The high level architecture of the project design is shown in Figure 6. In terms of a general overview, the architectural view is of a distributed simulation environment communicating RESTfully in over an intranet. A RESTful Collaborative Node (RCN) and a RESTful Collaborative Model (RCM), interface, respectively with a model running in a DEVS-Suite simulator. The layered structure within each node signifies the flow of data from the model to the RCN or RCM to the REST Handler onto the intranet returning in a reverse fashion through the appropriate node based on the URI indicated in the Http request or response. As the architecture depicts, the overall system model is comprised of three component models, one under the RCM node and one in each RCN node. The overall system model design and the model composition for the project are described in the Model Development section.

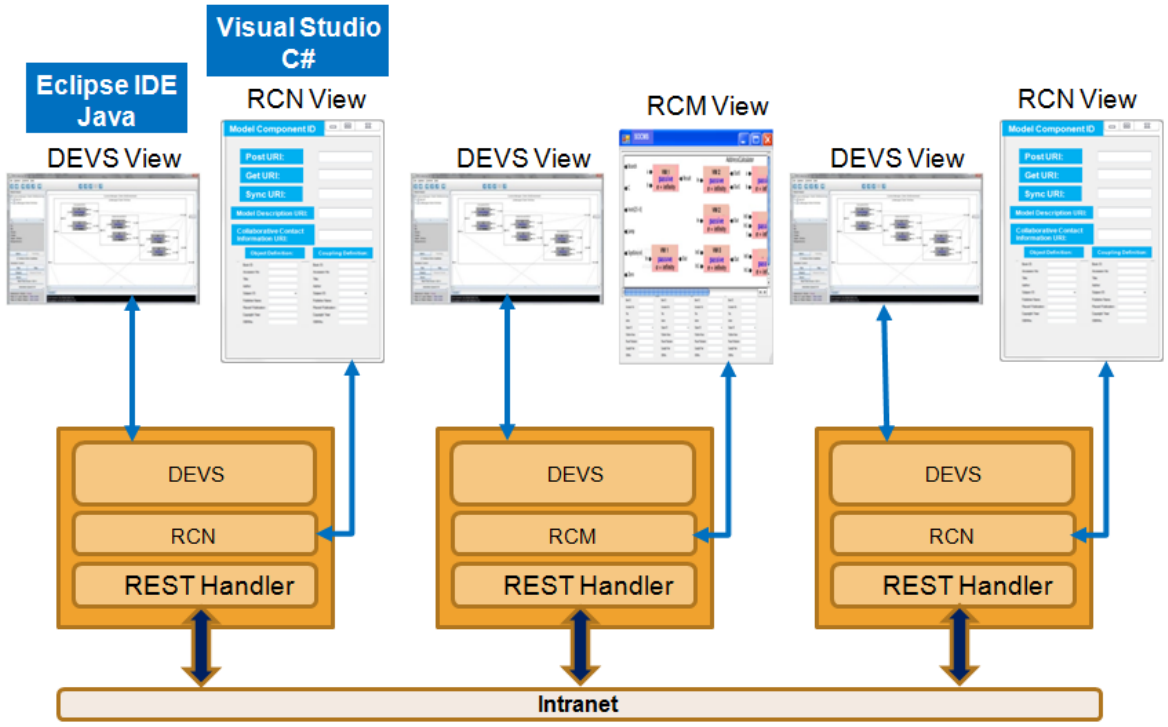

Figure 6 – High Level RCMS Architecture

The project's architectural approach is enhanced by the RCM/RCN middleware design providing plug-and-play interoperability employing a RISE like methodology [10]. Relying on a RESTful style for communication provides a lightweight interoperable approach using resource definitions with low overhead and a messaging interface absent of a required standard (clear text or JSON constructs are followed as much as possible). The nature of data flow in the system model is from the model through the RCM model and back for distribution to other RCN nodes as dictated by the coupling logic selected by the RCM modeler in the RCM View.

Figure 6 shows the views available to the user in this project. The DEVS View supports the model development for the respective node. The RCN view supports the model configuration, model description, input/output definition, URI definitions, and supports RCN modeler –to-modeler inter-

communication. The RCM view supports the model configuration, model description, input/output definition, URI definitions, RCN-modeler to RCM-modeler inter-communication., the coupling definition for all components, the I/O and status display of all components, the state of the collaborative models, and result and debug files received from RCN nodes.

REST and resource orientation in general use HTTP to exchange resources formatted with common content types, such as PDF, XML, HTML and JSON [11]. For this project, wherever possible the JSON format will be used in the http body. The REST approach in this project exposes the model interactions as resources using uniform Http methods to transfer messages to/from resources. The resources exposed in this projects' approach are principally associated with content passing between RCN and RCM, and MVC *actions* associated with the interoperability of a *conservative*[8] synchronization approach exercised by the RCM on each node model in the overall system model.

Each resource identified in this project coincides with an active method managed by the RCM and RCN partitions respectively. Each RCM/RCN partition consist a MVC pattern in the design. The RCM or RCN controller, coupled with the REST Handler, which is part of the ASP.NET MVC Framework, respond to the Http requests and generate the Http responses accordingly. Those interactions are responsible for the views presented and the model activity within the DEVS simulation environment, and for the transfer of data to/from nodes.

Resources used in the communication between RCN and RCM in this project follow the URI definitions shown in Figure 7. This resource structure coincides with RISE template structure and forms the basis of the http requests and responses for the project. The simplified resource structure shown

[8] The distributed simulation approach is usually composed of a number of sequential simulations referred to as logical processes. Synchronization is paramount in that process and required to consistently produce the same results as if executed in a single processor. The conservative approach to synchronization satisfies local causality constraint through ensuring safe timestamp-ordered processing of simulation events within each logical process.

reflects the nodal representation of the project architecture. More robust architectures would utilize an expanded structure as exemplified in Al-Zoubi and Wainer's paper [10]. The "{Controller}/{Action}" URI extensions shown in the resource structure support a conservative synchronization approach with the "action" segment in the http request protocol specifying either *content* messages {content} or *synchronization* messages {sync}. (Note that the complete list of URI resource extensions for this project is defined in Table 1.)

The controller/action definitions are similar in nature to the conservative designations in that the content messages consist of the Output messages referenced as "Y" and External (Input) messages referenced as "X", both of which are converted appropriately in the façade for transfer to/from the model. For this project no transformations (e.g. meters to feet) are conducted on the Input/Output messages. Synchronization messages include messages to Initialize ("I") or start the initial simulation phase for each component, a message to "step" forward in simulation time which results in execution of the Internal ("*") component's transition phase and in the Collection ("@") of output data, and a "stop" message to signify to the model that the execution of the simulation is complete. Note that the Done ("D") message, which marks a component's phase completion in the conservative approach, is not supported. Interruptions in the model execution cycle are allowed and processed through the "pause" signal.

Figure 7 - RESTful Collaborative Model/Node Resource Structure

As shown in Figure 7, the resource structure used in the project compares to the RISE template shown in Figure 2; the “workspace” is either the RCM or RCN node; the “servicetype”, although meant to signify different services, is used here to identify the project’s implementation as a unique service; and the “framework” for the project is solely DEVS-Suite modeling. The resource designation further specifies “controller” and “action” identifiers supported through the MVC framework. As extensions to the template, they serve to further identify functionality associated with the RCM and RCN MVC middleware. These controller/action collections are further defined to represent *content* or *synchronization*, configuration, and inter-communication actions as exhibited in Table 1

Available to any external user from the RCM is the system model description accessible by going to the “…/rcm/ap/devs/Home/Index” resource. Note that the Internet Protocol (IP) address (designated by “ … “) within the resource URI, serves to segregate the nodes for communication. The text content provided in the Home/Index resource is in the modeler’s format and describes the overall system model supported by the node, the components included, the components coupling, and the input/output definitions. The intent of this resource is to allow an open forum for information, accessible to all modelers. Constraints, assumptions, version designations, and modeler contact information associated with the model, as well as the IP address of the node, a list of the URI designations supported by the model, and any additional URI designations with their definitions are included in the description.

Also in a textual format, is the “/Home/Results” resource to access a flat file for the results collected for the system model run, the “/Home/Debug” resource to access a flat file for debug/error information collected during execution, and the “/Home/Notes” resource provided to allow notes to be passed back and forth between modelers’ for communicating update activity, pending down times, changes to the model, and other information as needed.

Table 1 RCM/RCN URI Resource Extensions

Controller	Action	Method	Header	Body	Description
Home	Index	GET	JSON	name/value pairs (text)	Node returns the model configuration which describes the component, provides a definiton of input/output structure and type, time advance and initial state, as well as modeler information.
Home	Config	GET POST PUT UPDATE	JSON	name/value pairs (text)	Configuration view for RCN and RCN modeler's; provides display/entry of component descrition definition, message structure definition, synchronization definitions, etc.; data is persisted at the node based on version ID.
Sync	{sync ID}	POST		<empty>	The context of "sync ID" is to signify model execution activity to correspond with start, pause, step, and stop.
Content	Input Output	POST	JSON	name/value pairs (text)	Content passed as X (Input) or Y (Output) messages: RCM passes X; RCN passes Y.
Home	Results	GET	text/	ASCII text	Returns collection file for the completed simulation run.
Home	Debug	GET	text/	ASCII text	Returns log file for errors encountered during the simulation run.
Home	Notes	GET	html/text	ASCII text	Modeler notes passed in cleartext to the appropriate node (RCM or RCN component)

The RCN provides similar actions through the resource extensions identified in Table 1. However, the intent in the RCN node is to provide definition of the component (model) level of fidelity, synchronization parameters (t_a), resource identifiers, input message(s) and type, output message(s) and type, and debug/ result capabilities. The RCN action resource extension “Notes” provides a text based output in which the modeler can identify revision updates and changes incorporated while the system modeling effort is in progress,

The Content resource provides for data passing between nodes. The RCM passes input data to the appropriate node while the RCN nodes pass their respective output data to the RCM. Control associated with model execution is posted by the RCM based on the minimum time advance set in place before collaborative model execution starts. The definition of the sync IDs involved with that control is listed in Table 2 and is described later in this discussion.

Breaking the node structure down further, the interaction between the REST partition and the DEVS partition is shown in Figure 8. The REST partition and the DEVS partition are designed as MVC applications. Both the RCM and RCN are developed as C# applications in the Visual Studio 2012 IDE while the DEVS models are developed in Java using the Eclipse IDE. The two partitions are integrated through facades that incorporate the Common Language Runtime (CLR) to Java Virtual Machine (JVM) bridge shown in Figure 9. The bridge selected was the JNBridgePro[9] which provides support to develop proxies for shared-memory communication in which the JVM starts in the same process as the CLR or .NET application and communicate through a shared data structure.

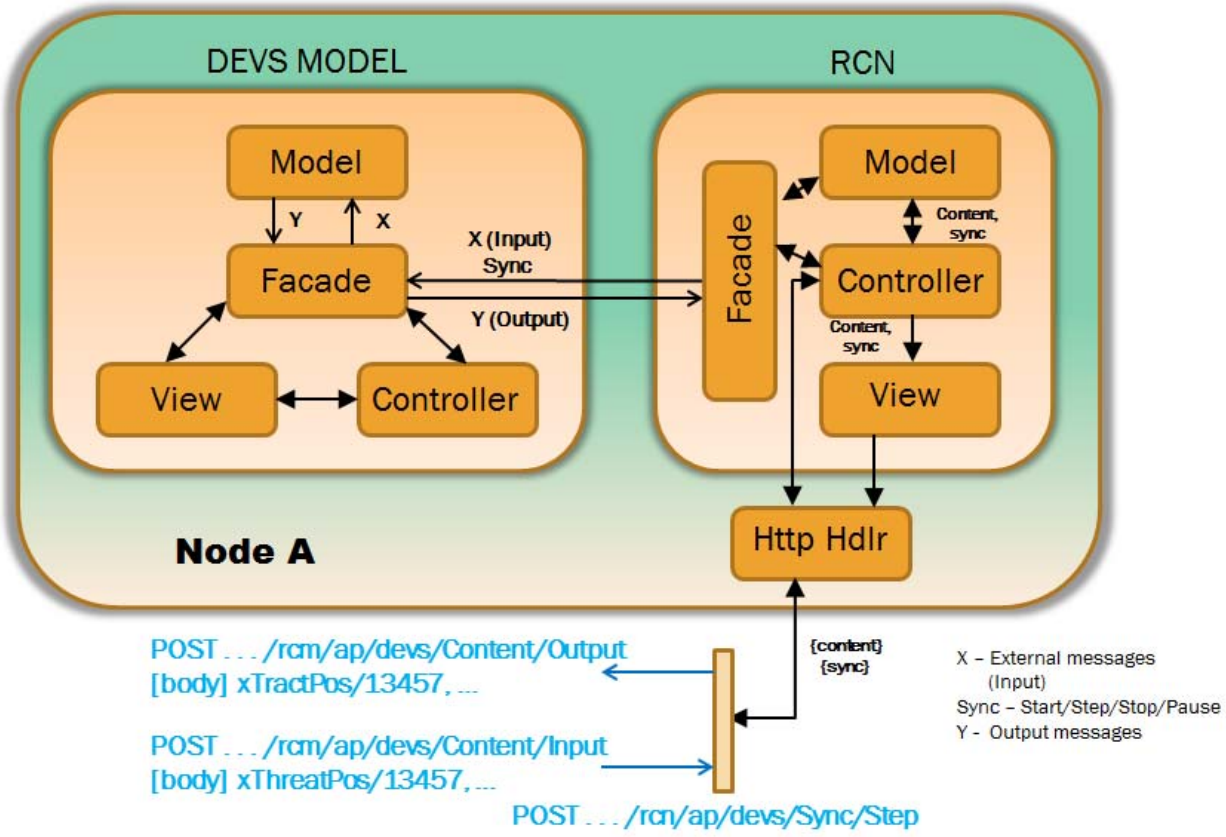

Figure 8 – High Level MVC Interconnects of Node A

[9] JNBridge, LLC, www.jnbridge.com, JNBridge is a registered trademark and JNBridgePro and the JNBridge logo are trademarks of JNBridge, LLC. JNBridgePro Version 7.0

The JNBridgePro software package develops proxy classes that provide the capability to use reflection to get public signatures for classes of .NET and Java. The Java Native Interface (JNI) is used to forward the call from the ,NET proxies to methods on the real Java objects while the JNI is used to register .NET implementations of native methods of Java proxies to forward calls to methods on the real .NET objects.

The facades are developed to comply with the input/output architecture of the node model as reported in the configuration data. The RCN façade provides proxies to exercise Java methods to move data to the DEVS input message bag (the Sync signal is part of the input data) and to generate the input event to the DEVS model through the DEVS *Controller.injectInputGesture* method. The RCN façade includes a C# class to send output data extracted from the model to the RCM. The DEVS pseudo façade includes proxies that exercise ".NET" methods to move output data objects from the model to the RCN environment. This façade is referred to as a pseudo façade in that it is contained within the model's Transducer class. All node facades employ the same functionality. Note also that the intent for the façades in this project is not to follow the full nature of the Model-Façade-View-Controller (MFVC) approach first introduced in the [18) and expanded later in [21].

The proxies generated on the .Net C# side represent a one-to-one class mapping to its Java counterpart and is responsible for managing the communication with that Java class. The JNBridgePro generates the proxy for the defined Java class and all supporting classes, parameter and return value types, exceptions, interfaces and super classes and formats the result into a Dynamic Link Library (DLL) file.

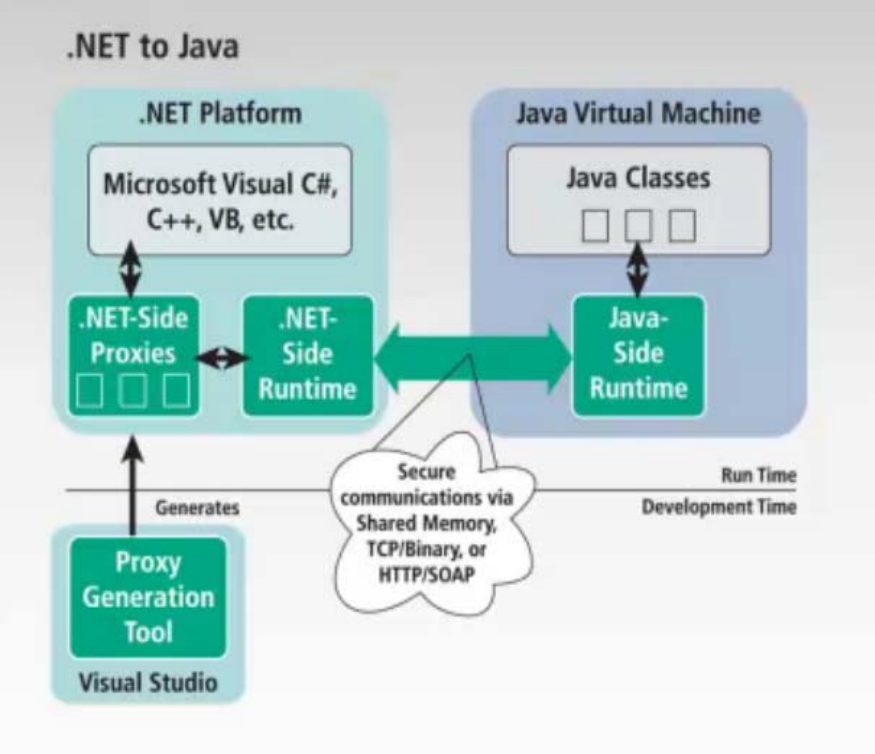

Figure 9 - Java to .NET Bridge[10]

The projects system model behavior design approach places constraints on the modelers to add new functionality to the DEVS models to support synchronization. The attempt here was to keep the functionality simple and not disturb the DEVS formalism designed within the model. The key feature implemented, maintains the distributed models in sync by removing the conservative simulation constraint requiring (described in [10]) querying of the next simulation time followed by resubmission of the minimum time to each model. Instead, the approach in this project is to rely solely on the injection of a sync "signal" to excite model state transition.

As defined in the background section of this paper, the DEVS formalism state transitioning depends on the modeler's implementation within the external transition, internal transition, and output functions. To keep the distributed models in sync, the sync signal "step", generated by the RCM, is used

[10]JNBridge artifact found at http://www.jnbridge.com/jn/kb/?p=286

by the RCN façade to advance the model through its next time advance. The “step” sync ID is used to transition the model to proceed in a normal manner according to its design. The constraint on the model is that when the output function completes, the model is placed in the PASSIVE state. Within the model, the external transition function updates the “sigma” for the model to its designed value. Going forward, the logic employed is such that when a “step” input is received, the model must be in a PASSIVE state to transition a normal functioning state. This allows for the “step” sync to follow the minimum “tN” (next simulation time) being applied to all models thus removing the need to query each model for its “tN”. The impetus is placed on the collaborative “master” modeler to collect the sigma values as part of his/her composition phase and schedule the collaborative model to “step” appropriately to meet component model timing. The model does not transition to the PASSIVE state until it has processed an output and thus will remain in the active, or non-passive, state for its intended sigma, ignoring any “step” sync inputs received until the “tA” (time advance) has expired.

As identified in Table 2, a “normal” sync ID allows for the “model or node” modeler to operate independently for updating their model. It also allows, through additional constraints, support for coupled models within the distributed design by requiring an additionally defined input that allows daisy chaining the “step” sync through the coupled atomic models.

As identified in Table 2, this approach also supports the “pause” function in that delaying the “step” sync for all distributed models allows them to remain in the PASSIVE state until the next “step” sync occurs.

Table 2 Sync ID Definitions

Sync ID	Behavioral Description
start	Signal to indicate to the node to transition to the initial active state.
step	Signal to indicate to the node to transition from the passive state to the active state and set it's time advance to the nodes designated sigma.
pause	Signal to indicate to the node that the collaborative model execution is in the "pause" state.
stop	Signal to indicate to the node to stop model execution. This allows for taking single nodes out of the collaborative model execution.
normal	Not a distibuted "signal"; intended to be generated by the independent modeler for test and model development purposes.

The two major activities that deal with data flow in this project's design, RCM transfer of input parameters to the appropriate node and RCM retrieval of model output parameters, are shown in the sequence diagram depicted in Figure 10. The sequence identifies the data flow for one complete RCM cycle which starts from a scheduler configured by the RCM modeler. Designated inputs for the appropriate node are passed as a JSON input to the Content controller Input action on the RCN node. There the RCN controller converts the input data to the appropriate data type for this projects collaborative model implementation. The RCN Thread, through a proxy created by the JNBridgePro, inputs the data into the RCN Generator component of the DEVS experimental frame. The RCM then issues a "Step" sync signal to the RCN Sync controller. The RCN Thread, through the UserGesture proxy calls the method to step the DEVS model. The DEVS model then proceeds through its pre-defined time advance and executes its internal transition function which culminates with the output function passing the parametric data to the RCN Transducer for model output. The RCN Transducer then executes its Java proxy to transfer the data to the .Net side RCN Thread parameters where they are converted to strings

which ultimately become a JSON output. The RCM, through an access to the Content controller Output action returns the output data as a JSON string to be processed in the RCM Thread.

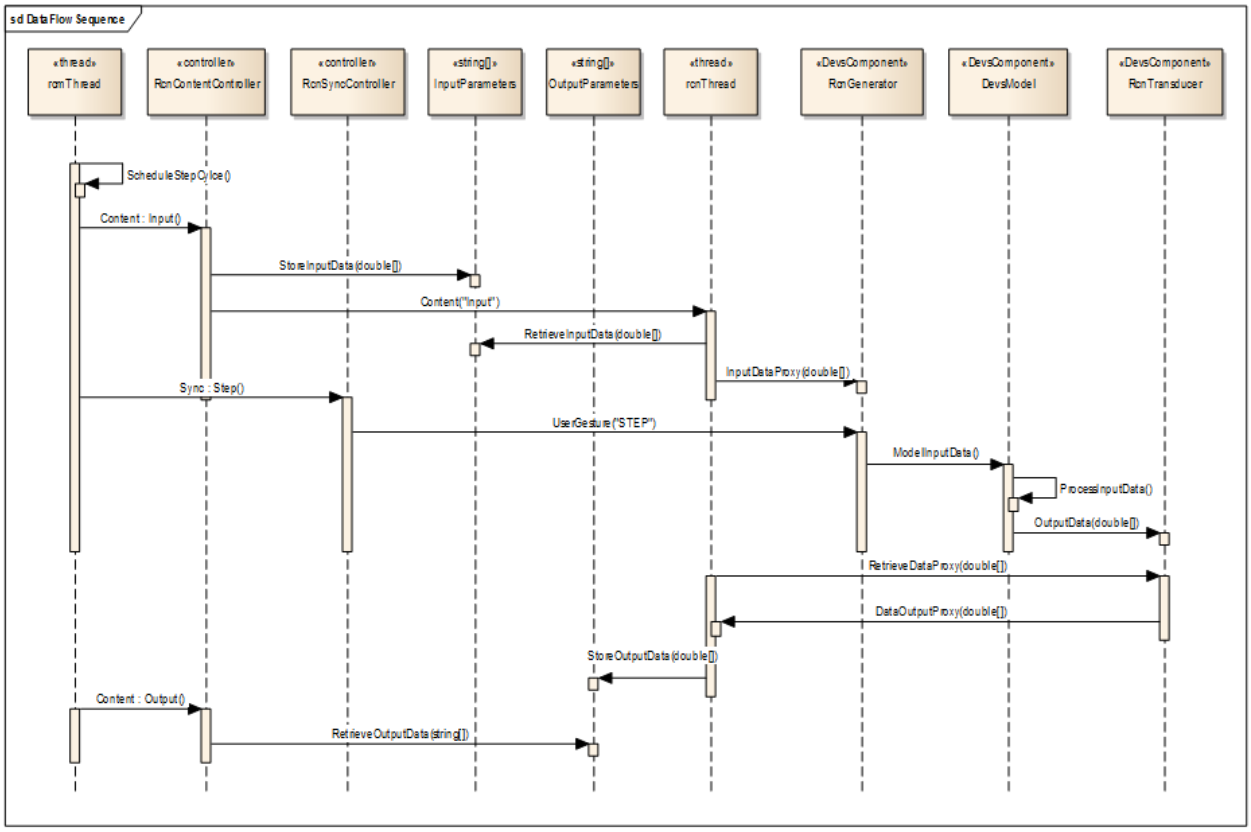

Figure 10 - Input/Output Data Flow Sequence

The RCM Thread is started through a local web access on the modelers' node to the Sync controller Start action. The instantiation of the thread constructors' is designed through the proxy interaction to start the DEVS-Suite simulator for that node and loads the appropriate model. As a constraint imposed through the bridge development with JNBRidge, an instantiation of the DEVS-Suite simulator must be started together with the JNBshare application on the model node prior to executing the Start action by the RCM Thread. The RCM Thread then starts the models on the appropriate nodes

through access to the node's Sync controller Start action. The view returned from the RCM Sync controller Start action provides the RCM View as shown in Figure 11. Control for the collaborative model, as well as individual node models, is provided.

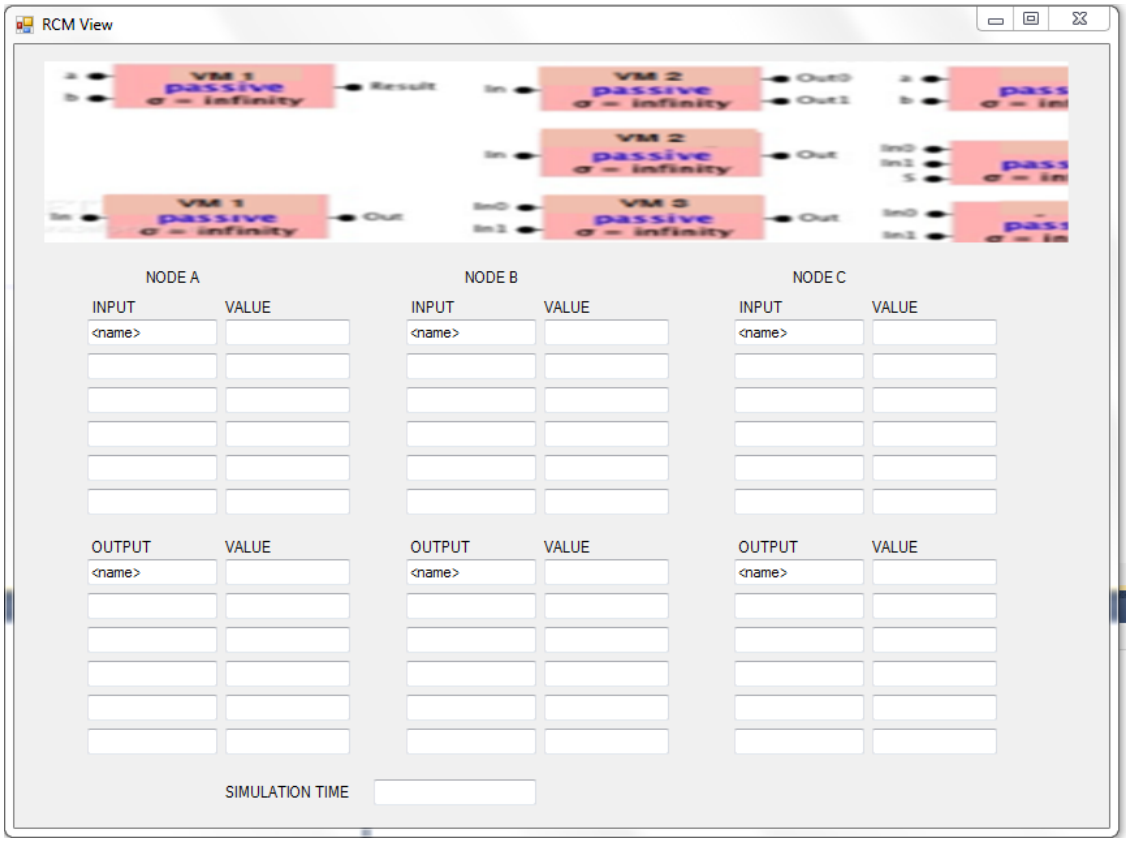

Figure 11 - RCM View (Prototype)

In a similar manner, the model start action selected from the RCM View, executes a process call to a sub-component that manages the system model synchronization and data flow management. The intent of the RCM View is to manage the system model execution and while the RCM View is alive, the sub-component will remain active.

As stated previously, the RCN View supports the model configuration, model description, input/output definition, URI definitions, and RCN modeler–to-modeler inter-communication. The RCM

View provides similar configuration functionality and also data-coupling selection for all components. component status and all inputs/outputs, the state of the collaborative model, and data collection (results) and debug files when requested from RCN nodes. Figures 12 through 13 provide prototype design views of the various windows designed for the project.

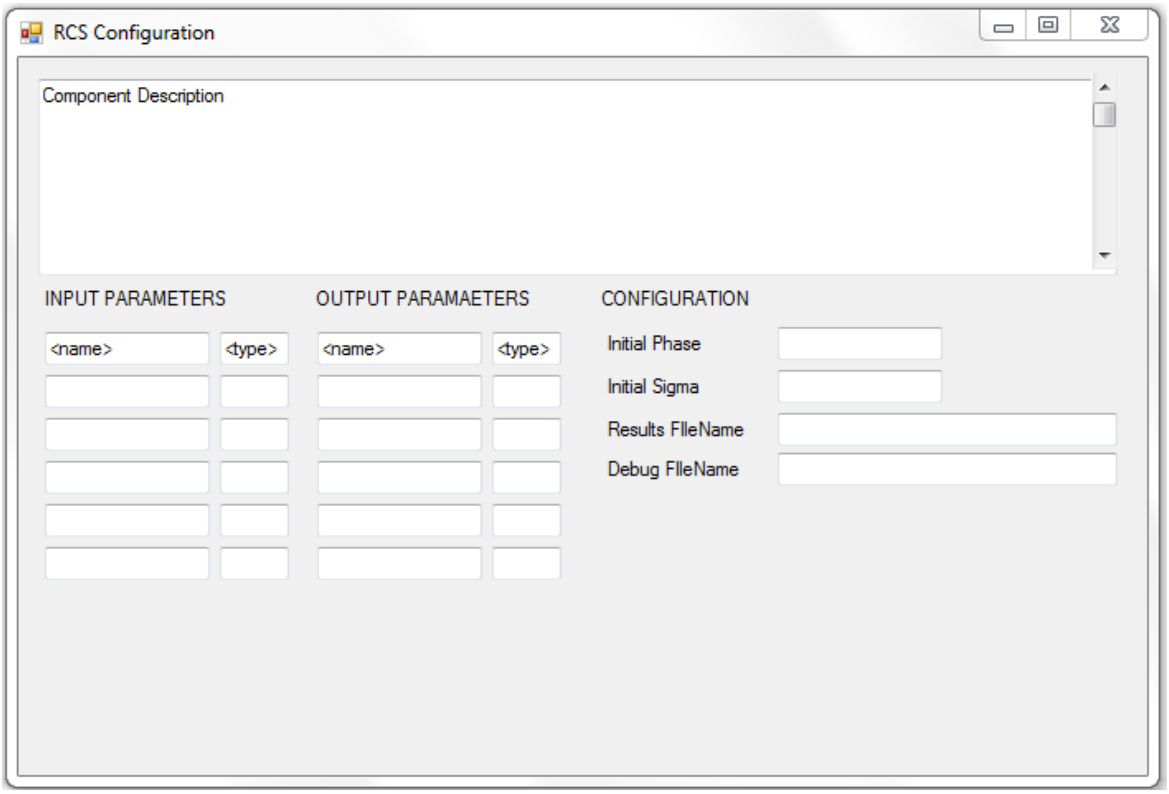

Figure 12 - RCN Configuration View (Prototype)

The RCN configuration view (Figure 12) provides entry for the model description supported by the RCN node modeler. It provides entry for the input and output parameters supported by the model and their types. It also provides entry for the models initial phase and the sigma (time advance) that the model executes. Entry for the filenames of the execution results and debug log is also provided. A

version number is displayed and signifies the ID under which the data is stored in the modelers' development database.

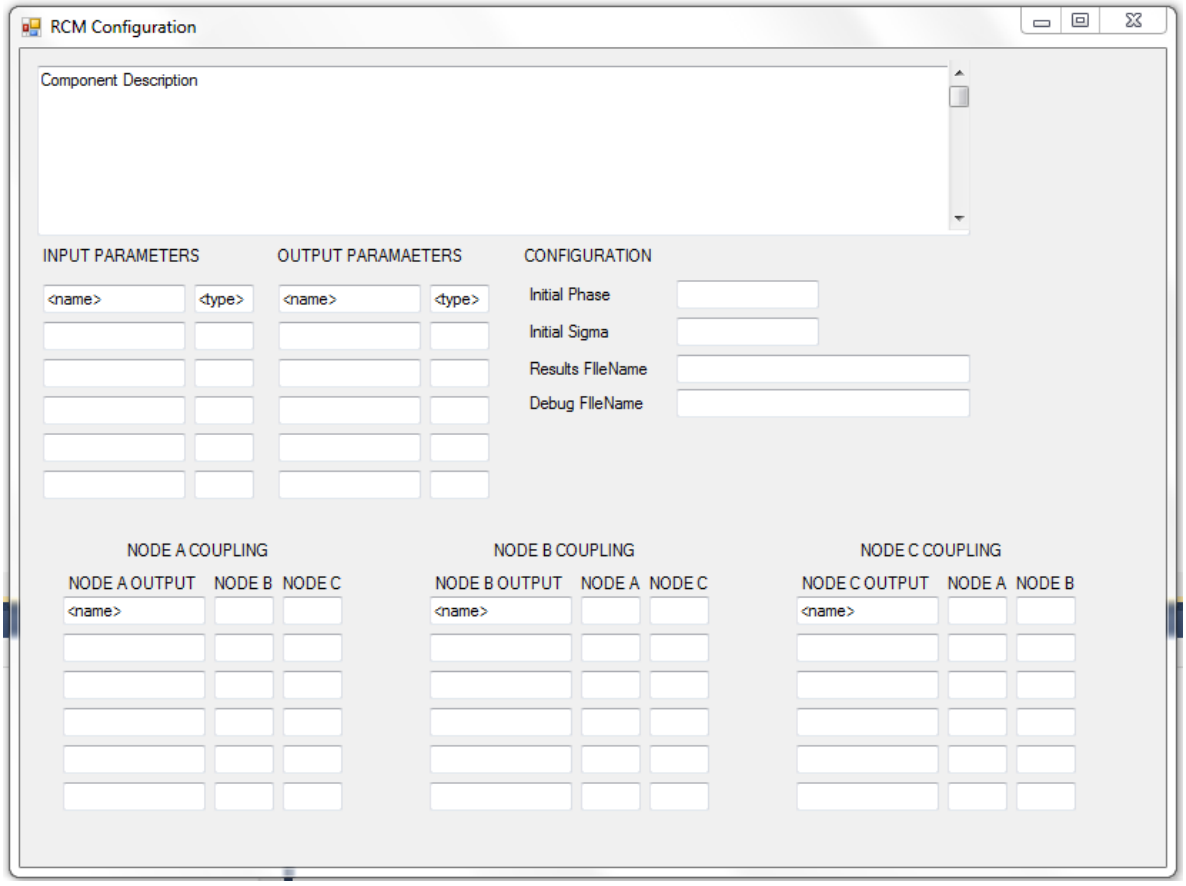

Figure 13 - RCM Configuration View (Prototype)

The RCM modeler, through access of each node's configuration, specifies the input to output coupling by checking the node box in the RCM Configuration View as shown in Figure 13. (The RCM modeler access the RCM Configuration View through a local web access.) The modeler specifies each node output that is provided by that node and couples that to the node input as designed in the collaborative model.

The execution of the collaborative model will be handled by a scheduler in the RCM Thread. The collaborative modeler will use an ISO (Input/Step/Output) map to assist in the definition of the logic that goes into the scheduler. The ISO map allows the collaborative modeler to account for variations in the time advance differences of the component(s) making up the node model. Table 3 defines the ISO map generated for the collaborative model used in this project. The map depicts two phases for model execution. The initial phase of model execution involves only the ThreatMotion model in Node B after which the simulated output of the threat position is sent to the Datalink model in Node C; an additional step in the Node C model is required to account for the delay simulated by the Datalink model before it generates its output. After the initial execution phase, the collaborative model will settle into a repetitive thirteen (13) cycle loop with the ThreatMotion (Node B) and UsvAlgorithm (Node A) models continually calculating their positions with a new threat position output by ThreatMotion every simulated 12 seconds accounting for the single step and output cycle in cycle 13 of the Datalink model. This execution will repeat until the Time To Interdiction criteria is met which will be returned to the RCM view and correspondingly stop model execution by stopping the RCM Thread.

Note that a cycle represents a time period over which the RCM Thread will execute all three functions on the nodes. A throttle will be incorporated into the design to allow the collaborative modeler to adjust the performance over the network.

Table 3 - ISO (Input/Step/Output) Map

ISO MAP (Initial Phase)			
Cycle	Input	Step	Output
1	B*	B	
2		B	
3		B	
4		B	
5		B	
6		B	
7		B	
8		B	
9		B	
10		B	
11		B	
12		B	
13		B	B
14	A	A	
15		A	A

ISO MAP (Repetative Phase)			
Cycle	Input	Step	Output
16	C	BC	C
17	B	BC	C
18	B	BC	C
19	B	BC	C
20	B	BC	C
21	B	BC	C
22	B	BC	C
23	B	BC	C
24	B	BC	C
25	B	BC	C
26	B	BC	C
27	B	BC	BC
28	BA	BAC	
29		A	A

The software configuration for the project design is depicted in Figure 14. The RCMS system developed in C# software language relies on the MVC Framework in Visual Studio ASP.NET 4.5 for a major portion of the functions provided and were not intended to be included in the software configuration shown in Figure 14. Also, the DEVS Suite simulation provides functionality to support the DEVS models utilized in this project; no attempt is made to define the class structure or interaction within that environment[11]. Note also that reference to Appendix B provides a comprehensive list of project software incorporated.

The software components in Figure 14 in tan color are C# classes, those in green are Java classes, while those in blue are the product of the JNBridgePro software tool and represent a dynamic linked library. In true MVC fashion, the software components standalone with no composition or

[11] Reference to [21] provides some insight to class structure and interaction.

inheritance and interact only to accomplish the task at hand. Not shown in the configuration is the class support for the views which interact only with the controllers to render the RCN configuration views and the RCM views.

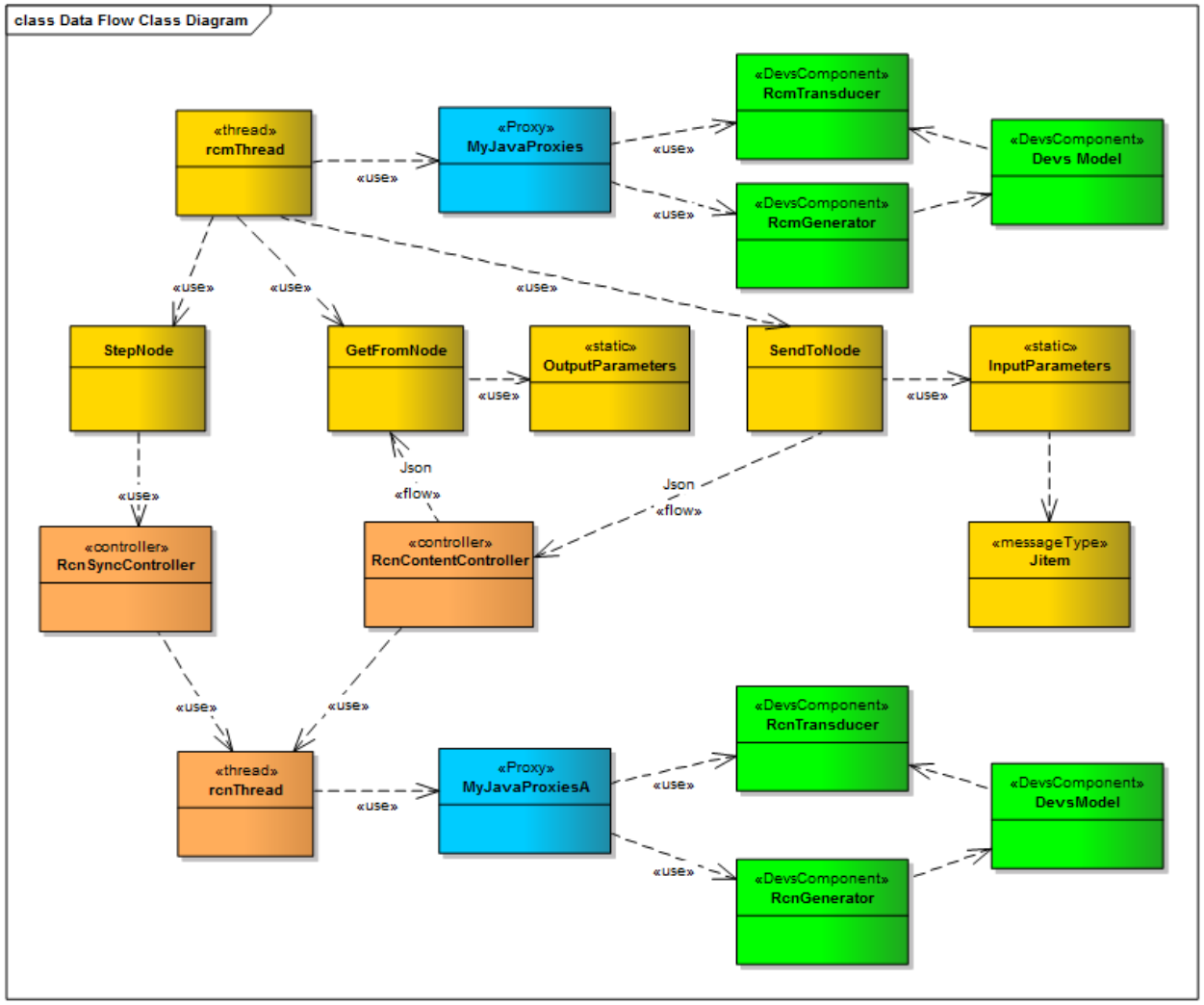

Figure 14 - RCN/RCM Software Configuration

MODEL DEVELOPMENT

The component models utilized for this project were taken from a previous DEVS project submittal[12] on Modeling Interdiction Methods in DEVS-Suite. These models were chosen to reduce development time and to reduce debugging necessary since these models were fully developed and functional in the DEVS-Suite environment. Additionally, this supports the architecture's suitability toward redesign and code re-factoring for incorporating capabilities into a new model. The model architecture from the Modeling Interdiction Methods project is shown in Figure 15.

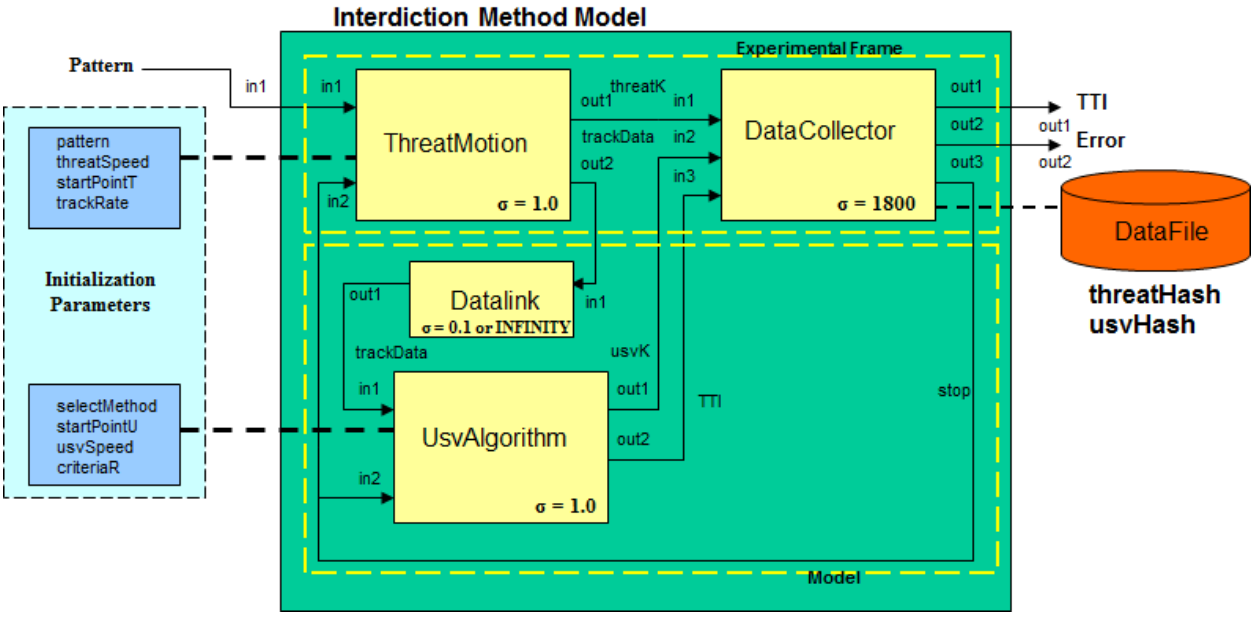

Figure 15 - Interdiction Method Model Architecture

For this project, the models were segregated into the RCMS architecture as shown in Figure 16. Each of the components reacts according to the component requirements initially generated and modified here to incorporate this project's collaborative modeling conservative approach.

[12] Heiser, J., "Modeling Interdiction Methods", 5 May 2009, CSE561 Course Project [Paper is available upon request.]

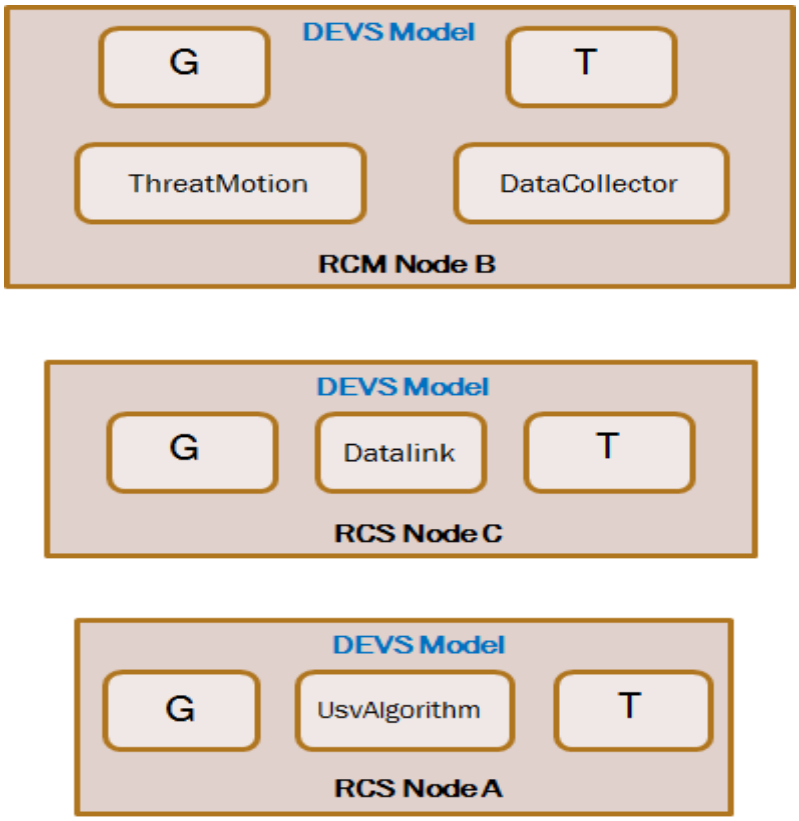

Figure 16 – Segregating the Interdiction Method Model to RCMS Nodes

Node B represents the RCM component of the collaborative model. The Interdiction Method model incorporated into this node is the ThreatMotion and Data Collection components. To minimize the modifications to the DEVS software, the Pattern input was eliminated and the Zero Order dead reckoning method and the A1 pattern were selected for initialization to the model. No modifications were made to the model requirements and the models' ThreatMotion and DataCollection components remained intact and can be referenced in section 3 of [16]. The resulting modified DEVS model architecture for Node B is shown in Figure 17.

To keep with the DEVS Suite simulation of an Experimental Frame, a Generator (G) and Transducer (T) components are incorporated. The principal inputs to the Generator are then the Sync

signal (generated locally by the RCM façade), the Time to Interdiction, and the Usv position. Although the UsvAlgorithm model in Node A generates kinematics (position, velocity, and acceleration components) only the "x, y" position components are shown to minimize clutter in the diagram. The output from the model through the Transducer consists of the position of the threat (trackData) (again only showing the "x, y" position data) as reported by the model every 12 seconds. The DataCollection component of the model is solely responsible for collecting and storing data to disk and for providing the final Time to Interdiction and position error for display. The "in2" input was handled by logic in the façade supporting the Sync signal "stop".

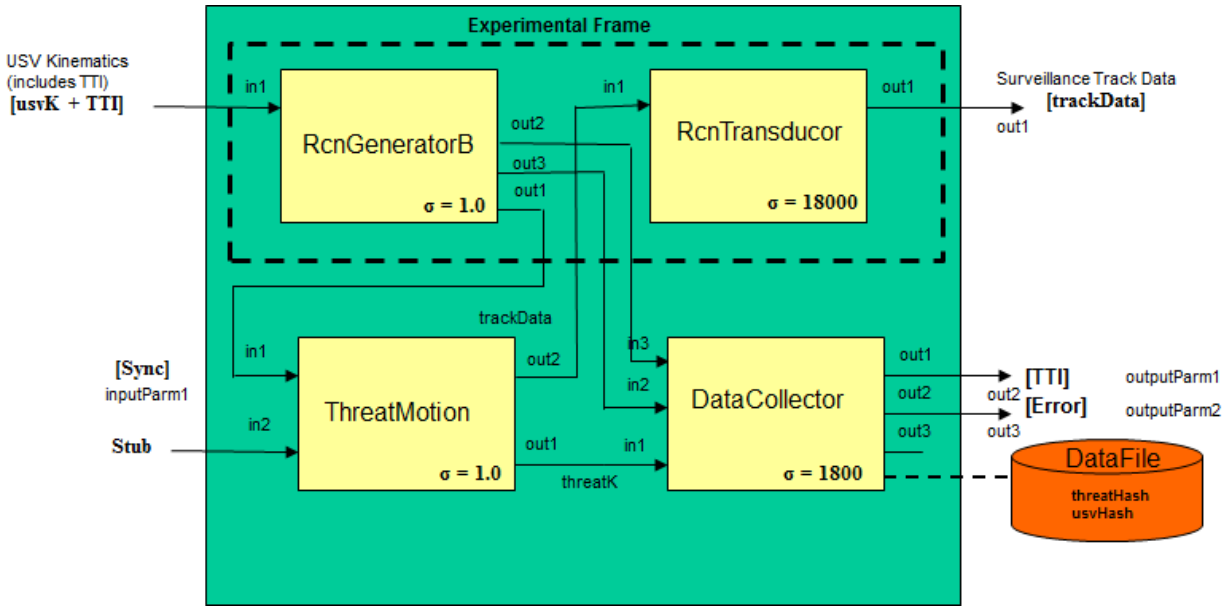

Figure 17 - RCM ThreatMotion Model Node B

The component from the Interdiction Method Model incorporated into RCN Node C is the DataLink component. An Experimental Frame, Generator (G) and Transducer (T) components are also

incorporated into the node. The Generator serves to accept both the Sync input and “trackData” input from Node B, and pass the data accordingly to the DataLink model. The DataLink model acts as a transmission medium from the simulated surveillance aircraft output from the ThreatMotion model and the simulated Usv reception. The model generates essentially a delay from the output of the ThreatMotion model to the UsvAlgorithm model. Its sole output is a pass through of the “trackData”. This is collected as an input to the Transducer and is generated as an output of the DEVS simulation executing on this node. The model architecture is shown in Figure 18. Although the DataLink model in the original Interdiction Method model did not require a “stop” input, to be consistent with the design in this project, a Sync signal input to the Generator will stop model execution.

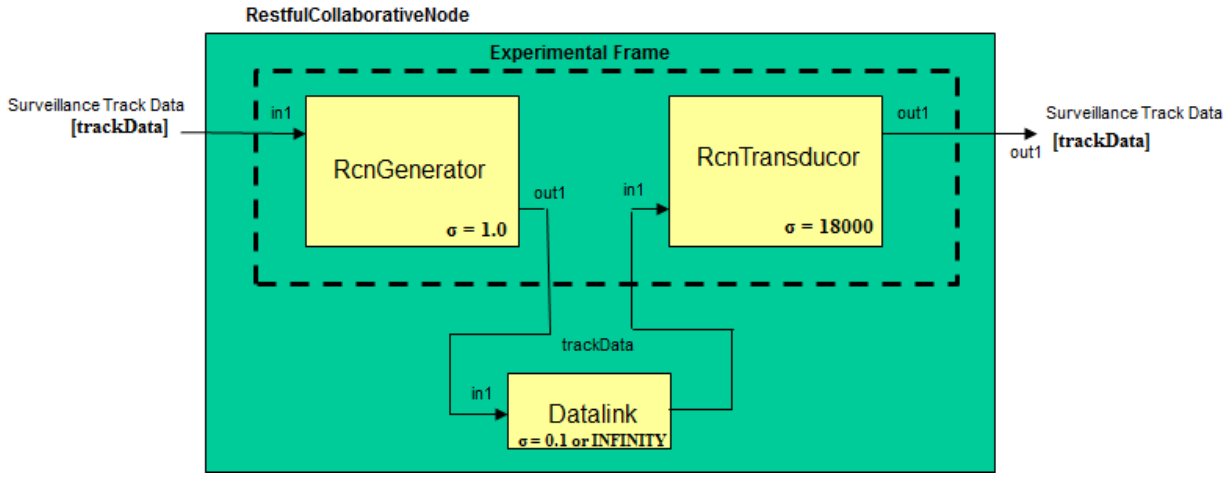

Figure 18 - RCN DataLink Model Node A

The component from the Interdiction Method Model incorporated into RCN Node A is the UsvAlgorithm component; the model architecture is shown in Figure 19. Similar to Node C model design, Experimental Frame Generator (G) and Transducer (T) components are incorporated. The Generator

serves to accept both the Sync input and “trackData” input from Node C, and pass the data accordingly to the UsvAlgorithm model. The UsvAlorithm model generates the position update for the Usv based on the “tractData” received and the dead reckoning pattern selected. The output from the UsvAlroithm model is the incremental Time to Interdiction calculated and the current Usv “x, y” position. The output is collected as an input to the Transducer and is generated as an output of the DEVS simulation executing on this node and passed to Node B.

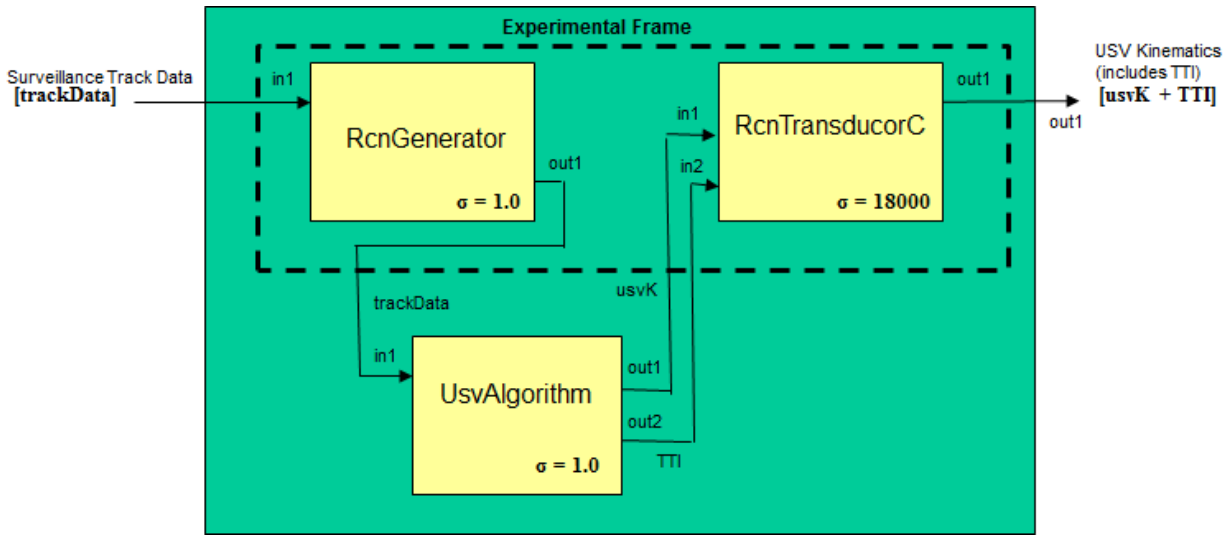

Figure 19 - RCN UsvAlgorithm Model Node C

In each of the RCN nodes, the Generator contains the logic to support the Sync signal input. The function of the Sync signal is to transition the model from a Passive state to an “Active” state. If the model has already transitioned to the Active state when a Sync signal is received, the input will be ignored. Once the model generates its output, it transitions to the Passive state awaiting a Sync signal input. The DEVS model in the RCM node works in a similar manner. Also note that the Generator and

Transducer components vary for each node to account for the variations in model input/output structure.

Principal interaction with the DEVS models is then through the Generator and Transducer components via proxies generated by the JNBridgePro software application. Refer to Appendix D for the Java packages and classes included in the process. Note that the Generator and Transducer names vary per node architecture.

EVALUATION

Working with MVC Framework and Entity Framework (EF) reduced the amount of development necessary to support “http” interactions and data manipulations. The Code First approach utilized with EF and outlined in [22] was challenging at first but proved beneficial after the learning curve. The underlying JSON support in the controller actions reduced the amount of code necessary to support model transactions and making the underlying http support transparent. Minimal code was required to manipulate the content data for JSON transport.

Database development although minimal within this project, was supported automatically by Entity Framework (EF). The class for the *model* configuration data was mapped directly to a table (automatically generated) in the database through the simple context class as shown in the code snippet below:

```
namespace rcn.Models
{
    public class EFDbContext : DbContext

    {
        //  Model class name goes here
        //                V    Table name goes here
        //                V          V
        public DbSet<ModelConfig> ModelConfigs { get; set; }
    }

}
```

Note that Entity Framework (EF) uses a pluralization service to infer database table names based on the class name of the *model*, i.e. in the code snippet above, *ModelConfig* gets pluralized to *ModelConfis.* A table with this pluralized name gets created and the table data elements and types are inferred from the *model*. The constraint to have the primary key in the data structure include the characters “Id” ensures that the table has an identity element included.

Initial investigation into the Common Language Runtime (CLR) to Java Virtual Machine (JVM) bridge *jni4net*[13] was abandoned due to lack of supporting documentation and complete examples. The documentation provided with the software downloads was sparse and incomplete. Blogs and Google searches proved less than informative.

As an alternative, the JNBridgePro software tool was selected to replace the jni4net bridge for proxy generation. The proxy software supports the interaction between the RCN node C# software and the DEVS-Suite model Java software. Instructions for building the proxy file using the JNBridgePro tool can be found in Appendix D. Note that for this project only the ".Net to Java side" option was selected; that is only calls within the C# software to methods and attributes within the Java software were exercised. The JNBridgePro tool does support building proxies to go from the Java side to the .Net side as well, and both proxy types can operate in the same environment. The proxy generation process results in the creation of a "DLL" file {MyJavaProxies.dll) which is added as a reference in the .Net project. Configuration of both the .Net side and the Java side is required to support the proxy and is outlined in Appendix E. To support the proxy configuration, software (*jnbcore.dll*) provided by JNBridge requires a property file to be generated. The class developed to support that file generation is included in Appendix F. Only two properties were configured in the properties file (*jnbcore.properties*); the communication protocol was set to TCP and the communication channel was set to port 8085. The remaining *jnbcore.properties* were left to their default values as noted[14] in the JNBridgePro Users' Guide [24]. Note that the JNBridge software is licensed protected and proper integration[15] of the license manager files is required for the proxies to operate.

[13] http://jni4net.sourceforge.net/ jni4net home page; bridge between Java and .NET (intraprocess, fast, object oriented, open-source)
[14] Refer to page 68 in reference [24]
[15] Refer to JNBridge knowledge base "Unable to load DLL rlm932_x86.dll . " http://www.jnbridge.com/jn/kb/?p=286

The configuration for the bridging mechanism selected in the .Net to Java direction lead to the code development as outlined in Appendix G. The Java code was created to develop a "main" class to fork separate threads to start both the *JNBCore* "main" and the *controller.Controller* "main". This was necessary to provide access to the proxies in the RCN Thread, when started, to instantiate the DEVS-Suite Controller. It also provided the means to start the DEVS-Suite node model manually[16] in a standalone fashion for testing purposes. Note that during collaborative model execution, this instantiation of the DEVS-suite Controller software from threading was not exercised for model execution; the instantiation from the RCN Thread start was utilized.

The RCN Thread was started through a call in the Application_Start method of the Global.asax.cs file. Activation of the Application_Start method occurred with the first access to the Sync/Start controller/action in each node by the RCM. As noted, this action instantiated the DEVS-Suite Controller and the intent was to load and start the DEVS-Suite model through software. However, for this project each node's model had to be loaded manually after the DEVS-Suite Controller was activated.

The RCN Thread, as well as the RCM Thread, principally accessed three methods inside the node's DEVS-Suite model: (1) a data input method *(altdeltext*) was developed and incorporated into the Generator class that directly input data into the model's input store parameters; (2) a method (*altout*) was developed and included within the Transducer class to allow extraction of model's output data; and (3) the `userGesture` *method* in the Controller class was called to step the simulation through the model's next time-advance. Reference to the code to access these methods can be seen in the C# code for the RCN façade in Appendix H. Java code development of the methods incorporated can be seen in the Generator and Transducer code in Appendix C. In addition, to support data output extraction, a "newdata" flag was created inside the Transducer to identify when data was submitted. The flag was

[16] Starting the thread software was accomplished by right-clicking on the "MyJavaThreads" class in the Eclipse Package Explorer browser, and then selecting Run as a Java Application

accessed in the RCN Thread to determine if a call to the Transducer's *altout* method was required to extract data.

The development of the DEVS models from the initial Interdiction Method Model resulted in the creation of a generic Experimental Frame that was applied to both the RCN and RCM nodes. This led to a reduced effort in model development, and supports the inclusion of future node development in the collaborative model environment. Configuration of the generic EFs was determined by the data input/output definitions of the node model. This was in keeping with the constraint of eliminating modification to the node model so as to maintain seamless operation from the original Interdiction Methods Model. Development of the node models for this project, resulted in no changes to any of the model components. The designated input and output ports were kept intact and data throughput maintained the same signature for the ThreatMotion, Datalink, and UsvAlgorithm components. The DataCollector maintained the same requirements for collecting both the USV and threat kinematics. Time-to-Interdiction (TTI) and the positional error were supplied to the RCM view to signify the end of the model execution in meeting the interdiction criteria as in the original model.

Figure 20 shows the RCN Node A DataLink model as represented in the DEVS Suite view. The view identifies the coupling between components, the current state of the component, and the current value of sigma (time advance). As noted the time advance for the Transducer is set to a large value to allow for extended model execution. Note that the RcnGenerator and RcnTransducer make up the Experimental Frame for the DEVS model. This same framework was applied to the model development for the RCN UsvAlgorithm model in Node C, and for the ThreatMotion model and the DataCollection components in Node B. As indicated in the EF component names depicted in Figures 21 and 22, minor modifications were incorporated to either the RcnGenertor, RcnTransducer, or both to account for variation in data port requirements of the node's model components.

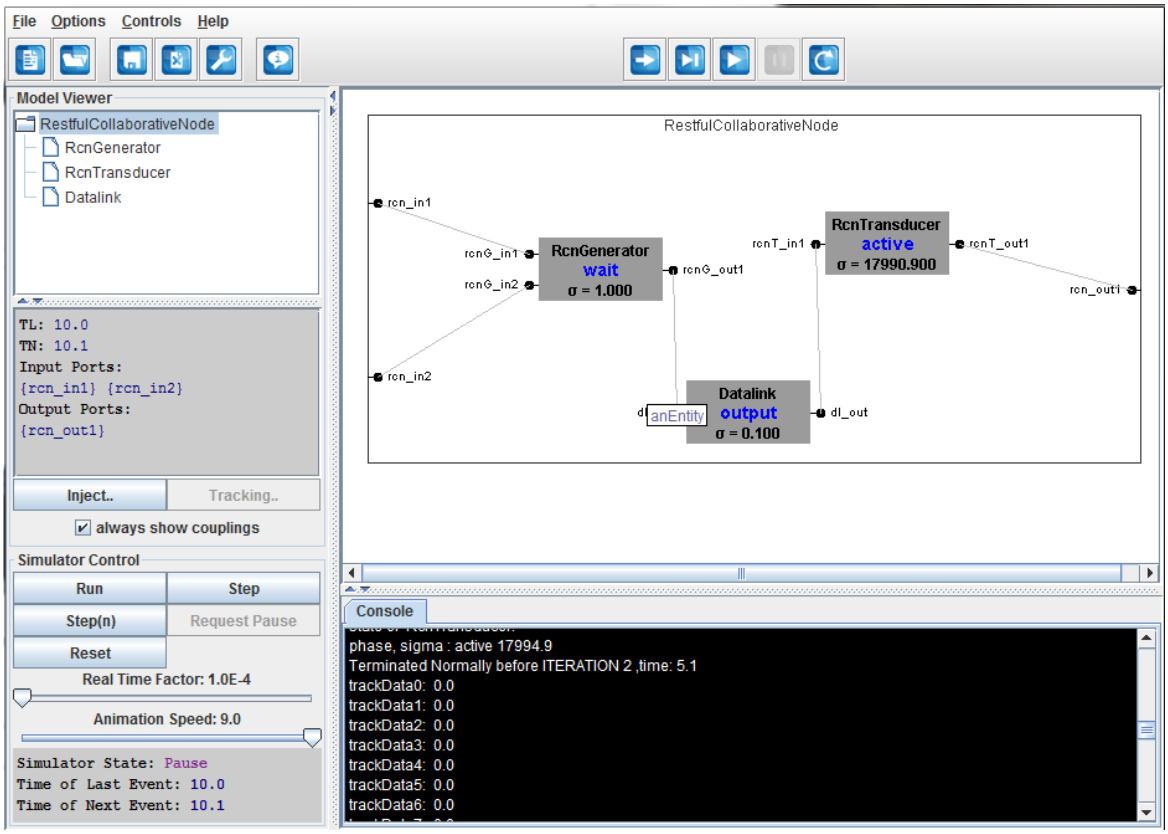

Figure 20 – DEVS View of RCN DataLink Model – Node A

The original intent was to input data into the model through the ***`injectInputGesture` method*** inside the Controller class. However, inability to access the inner parameters to support the method call led to the alternative data input approach using a method developed to input data to static members in the Generator class. This eliminated the need to pass the sync signal into the model and bypasses the logic to account for that signal. Note that the code for this logic was left intact within the RcnGenerator for future consideration and because the Generator component would not be transitioning through the normal input sequence.

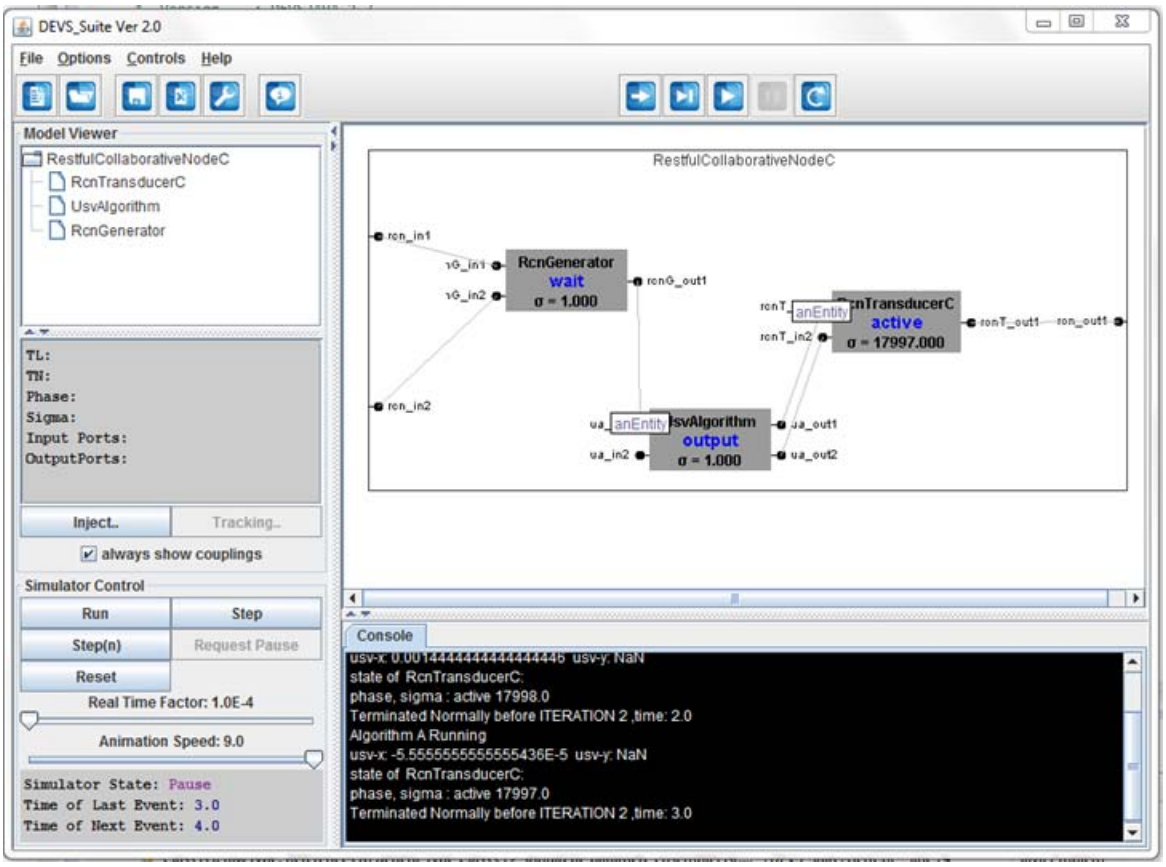

Figure 21 – DEVS View of RCN UsvAlgorithm Model – Node C

Data input from the RCM is passed to the controller action in JSON format. This required converting the string parameters into their corresponding data types for input to the model. Conversely, data output to the RCM node was converted into a JSON string as part of the post-back to the RCM Content controller Output action. The underlying JSON support was handled by the MVC Framework.

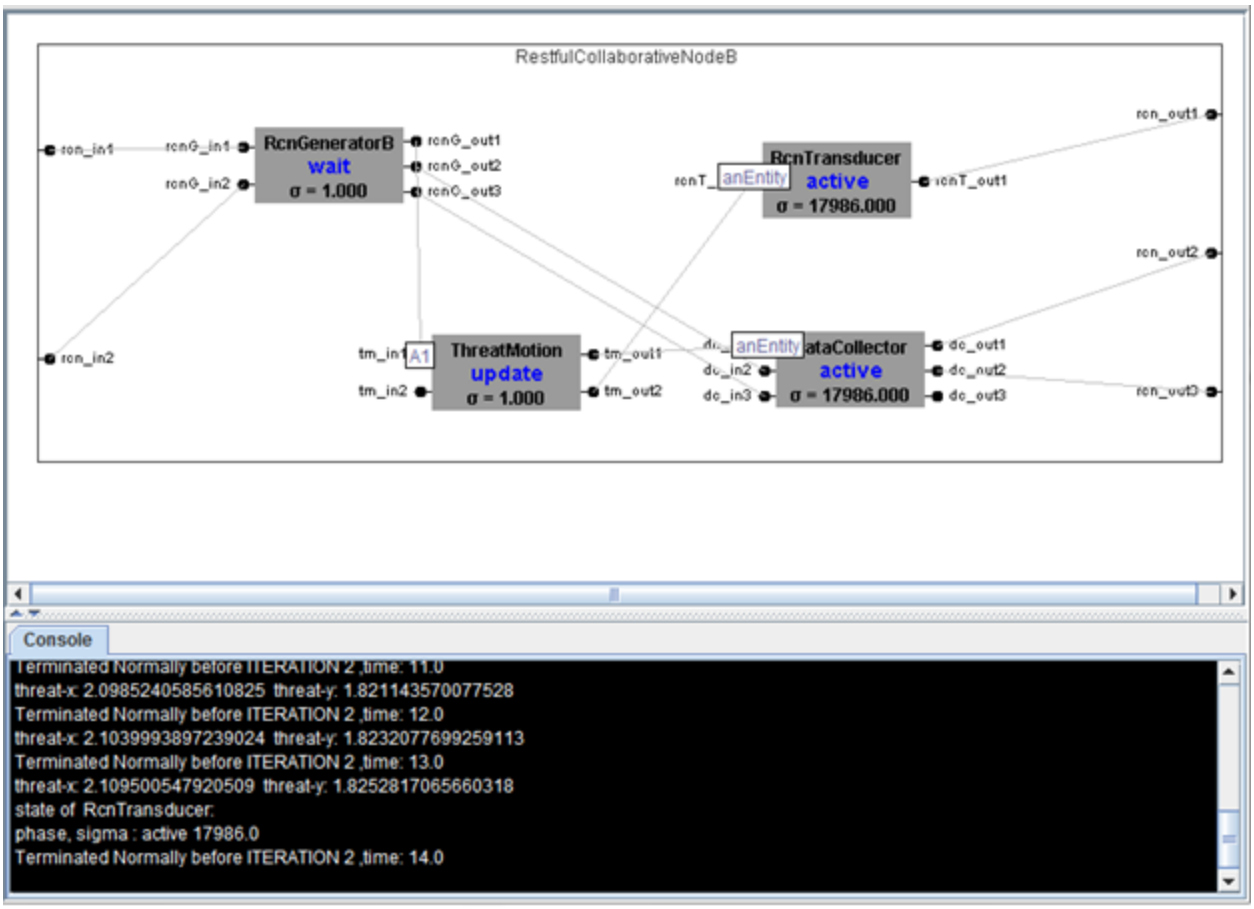

Figure 22- DEVS View (Partial) ThreatMotion and Data Collector Model – Node B

The ThreatMotion component was started with an “A1” designation indicating a straight line motion for the threat starting from position 1 (refer to the source ThreatMotion source code for the start position constant values for both the threat and the USV). The UsvAlgorithm was initialized to execute a Zero-Order dead reckoning algorithm.

Integration of the three models proved to be more demanding than envisioned. The hardware configuration involved three different computer platforms, each with their own idiosyncrasies . Debugging was more complicated due to insufficient tools to access information within the IIS Manager processes. Integration testing successfully completed the Initial Phase of execution and started the

Repetitive Phase of execution only to pause after completing data input and one step function to model A; the results are shown in the DEVS View screen captures from each platform in Figure 23.

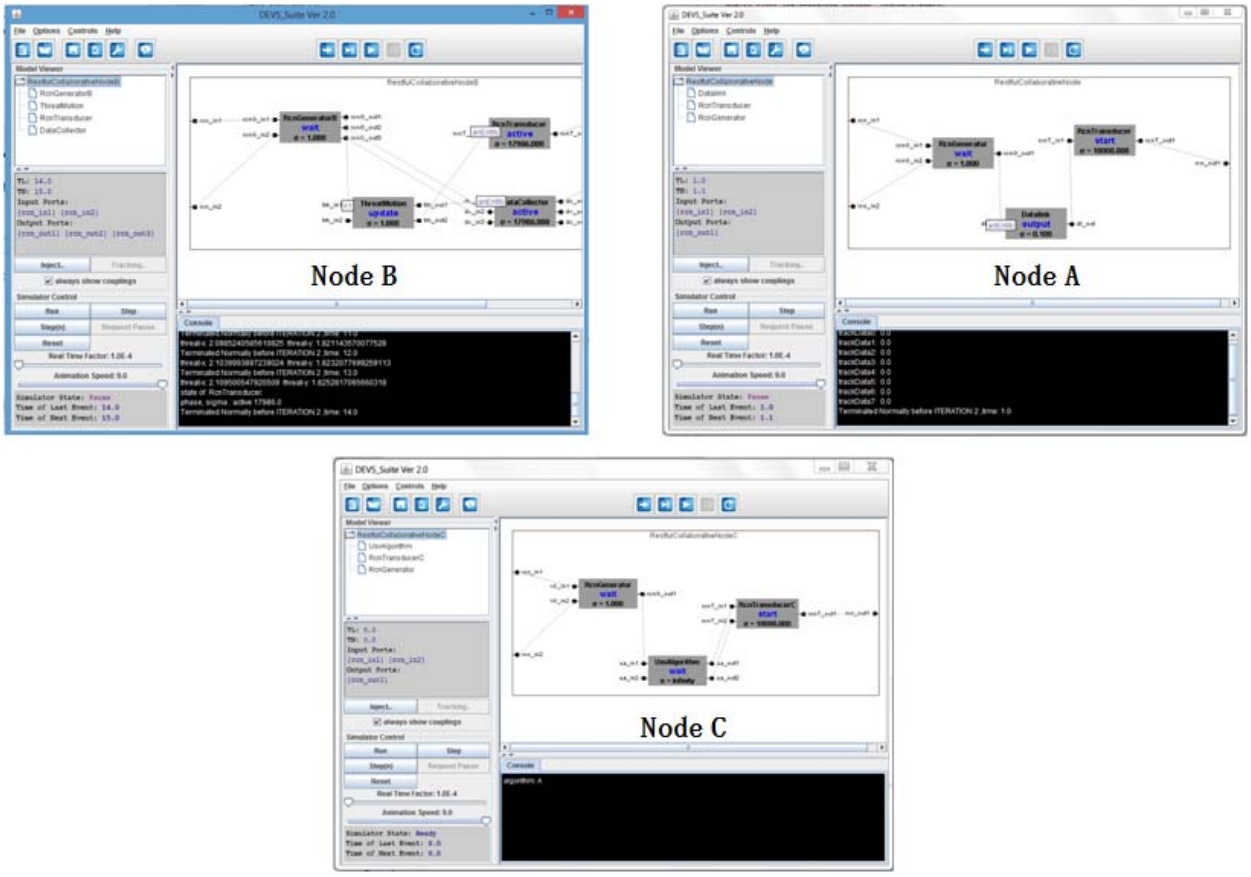

Figure 23 - DEVS Views from Integration Run

The views generated for an RCN node consist of a model data entry configuration view to allow the modeler to enter data associated with the node model component(s), shown in Figure 24, and a node model configuration data display to render the current, or latest, model configuration, as shown in Figure 25.

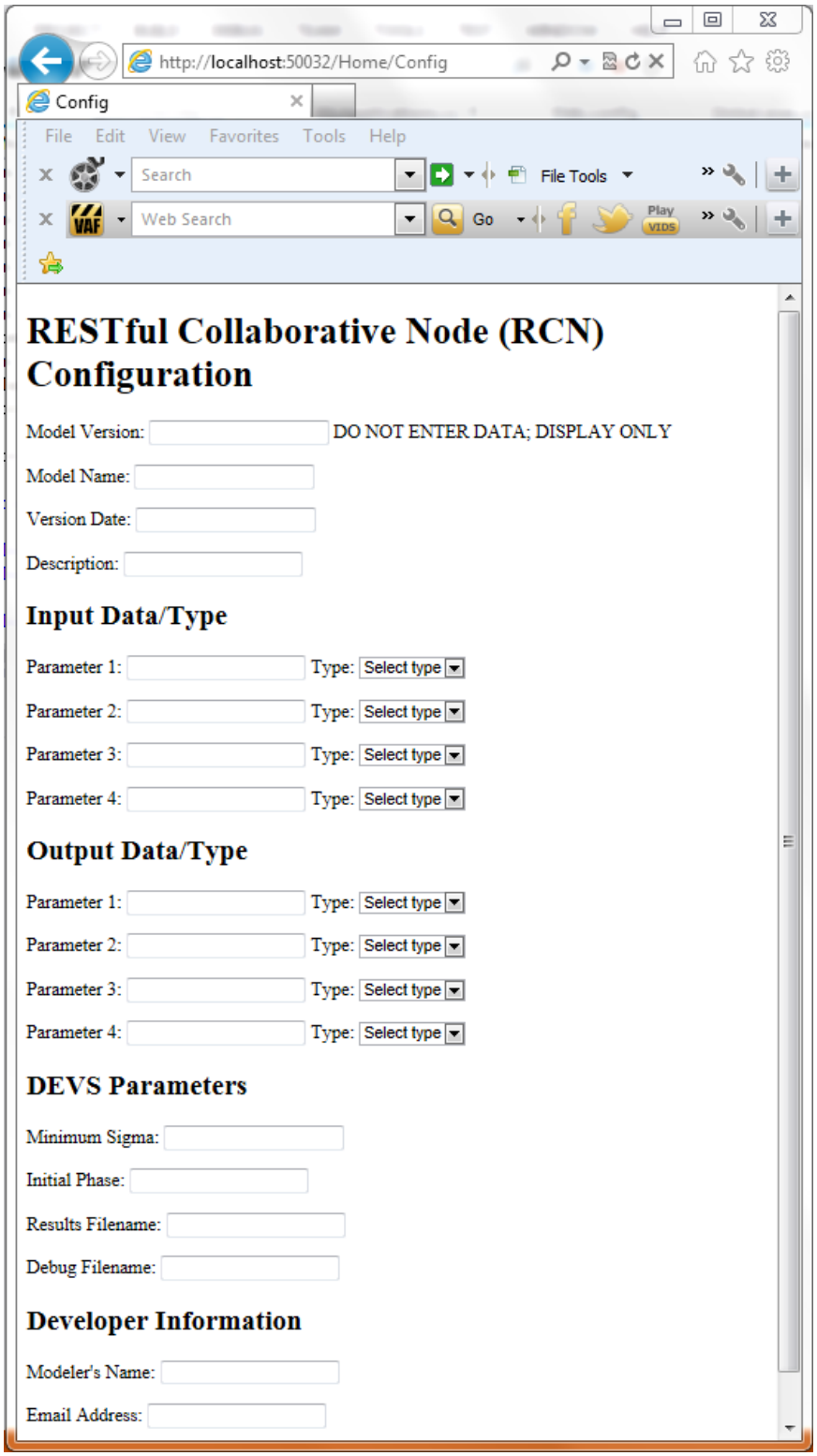

Figure 24 - RCN Configuration Data Entry View

The model configuration data is rendered through the view as shown in Figure 24. The data bound to the view is maintained in a database. For this project no functionality was provided to allow the user to search or select a particular configuration from those maintained in the database; a prospect for future consideration. The data entered is validated through the underlying functions of the MVC Framework and invalid data entry is not accepted. No login functionality was provided for the configuration data entry which inhibits remote users from entering data into the database; only the local user can access and edit database data. Therefore, vulnerabilities such as File System Function injection, SQL Injection, and illegal access to the database were not addressed. And since no “aspx” pages were generated, access to unpublished views were not a vulnerability to consider; views can only be rendered through a valid controller/action; all other accesses return a view depicting the message “Invalid Access: Please try again”.

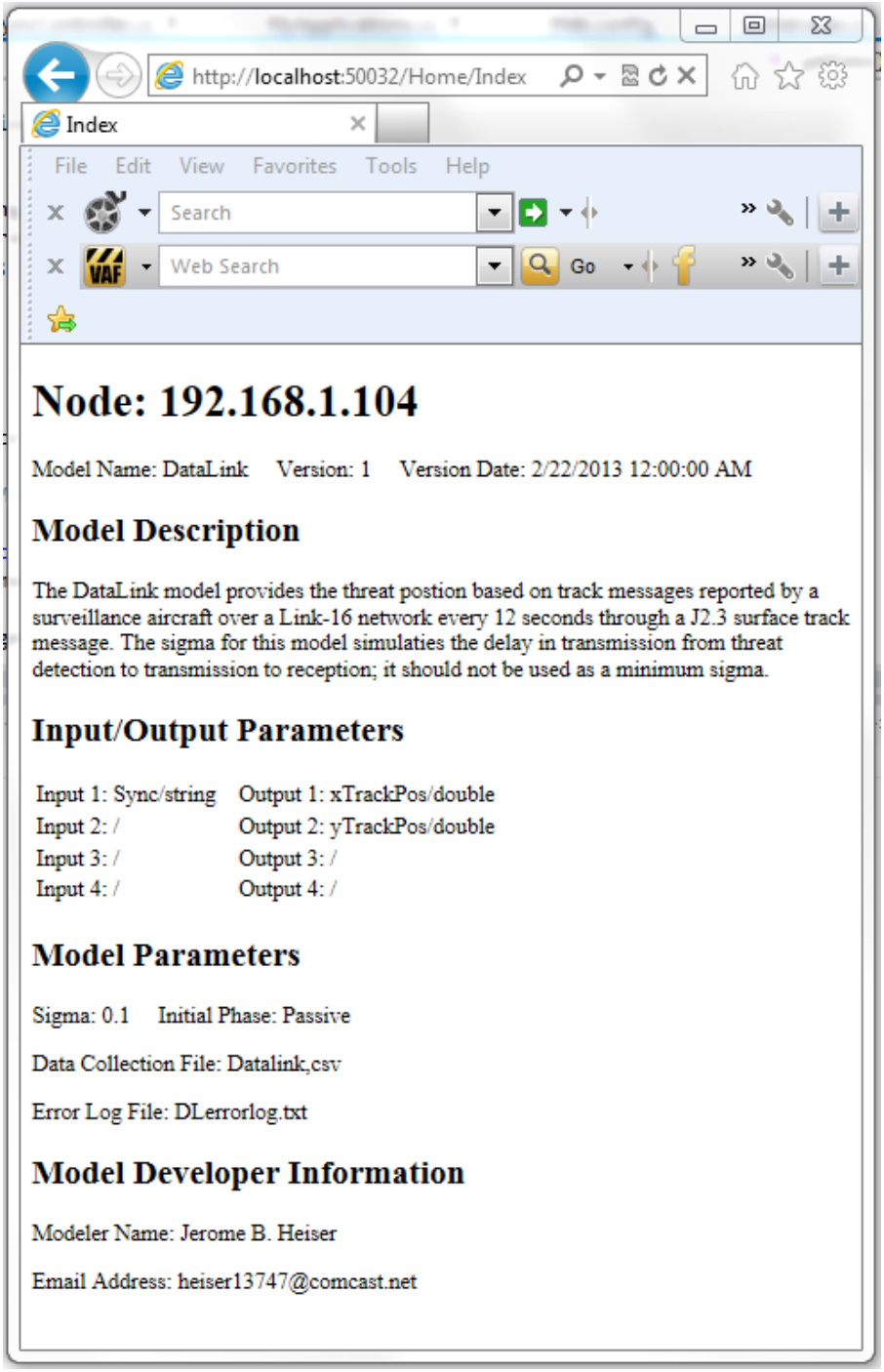

Figure 25 - RCN Configuration Data View

As the RCM Thread steps through the collaborative model execution, the RCM View shown in Figure 26 renders the input and output values for each node. The current node state is displayed to

indicate node model execution. The collaborative modeler can select the state for model execution as well as a specified throttle value that delays the extraction of output data from the node models after executing a step sync signal. The selected collaborative model state and the current cycle count are also displayed.

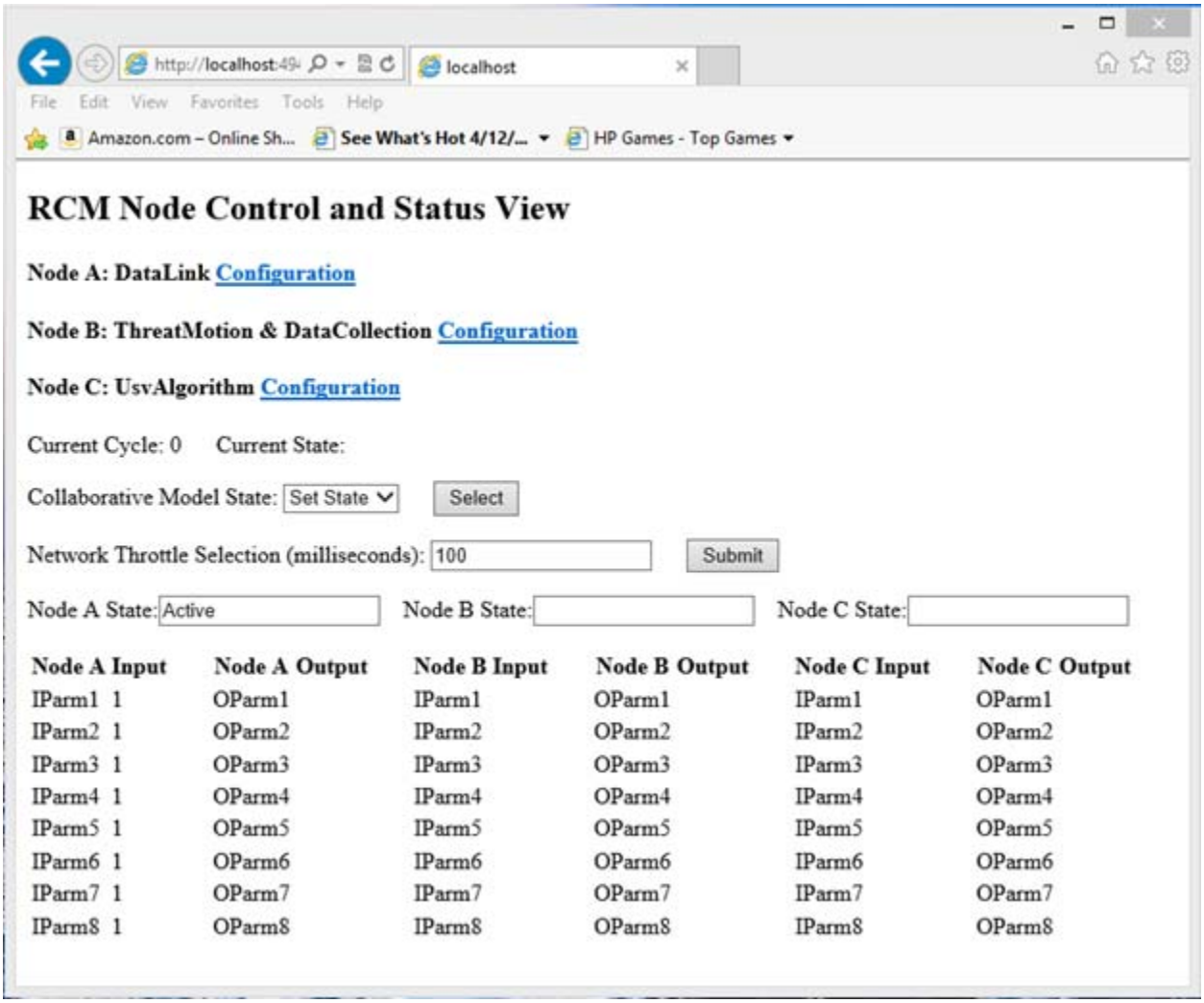

Figure 26 - RCM View

The RCM View provides access to the node configuration data; leaving the node configuration view redirects back to the RCM View. Access to each node logfile and error log is also provided.

CONCLUSION

The project objectives were successfully completed. Developing a RESTful collaborative architecture was accomplished through realizing methods that used Http protocols working together with the underlying REST functions of the MVC Framework to control model interaction. Establishing a functional interface through bridge software from controlling software to DEVS-Suite Simulation environment proved challenging but highly successful using the JNBridge bridge. The user interface, albeit rough, provided the functionality to create, persist, and allow access to node model configuration data, and to support control and status of the collaborative model.

The development of the RESTful distributed methodology for collaborative modeling proved challenging from the standpoint of software integration and interoperability. The initial bridging software selection (jni4net) was deemed non-usable and an alternative selection (JNBridge) was made which proved highly successful. Application interface development between the disparate computer platforms proved to be a debugging challenge and additional capabilities were required which ultimately resulted in the inability to complete the modeling integration.

The development of the user interface through the MVC Framework was noteworthy in minimizing effort required for view development; drag-and-drop controls to build frames was replaced with Html5 and CSS code; data binding was simplified by simply identifying the model associated with the view. However, the user interface views developed did not take aesthetics into consideration;. future work on developing CSS layouts that provide for more capabilities and enhanced interpretation/intuitiveness should be investigated; also consideration should be made to make the

views more pretty. Also, future work would look at a dynamic configuration function that could be made available to the master user to assist in the coupling of components.

Other areas of investigation throughout the project development needing future attention show that while the RESTful approach provides a quicker and easier way to instantiate Web services than the more traditional, SOAP-based/WS-* approach, lightweight REST stacks have difficulty accommodating sophisticated security requirements, such as:

- Authenticating/authorizing RESTful requesters in a uniform manner
- Integrating RESTful Web services with existing identity and access management infrastructure
- Monitoring and auditing access to RESTful Web services
- Enforcing service levels and quotas for RESTful Web services
- Propagating credentials across RESTful Web services, machine to machine

Investigation into DEVS/SOA [25][26] as a distributed simulation approach for the purpose of characteristic comparisons, and to further investigate the feasibility of cross-coupling approaches to extend alternatives , i.e. composition of RCMS as a RESTful service together with the DEVS/SOA web – based approach.

Future investigation into cross-browser, cross-platform application methodology such as supported by Microsoft's Silverlight can reduce the need for installation of .NET framework. Silverlight supports the MAC as well as Windows and is also supported in the Linux world through Moonlight.

Additional features such as a "reset" sync ID function can be explored in future efforts. The reset function would look at resetting the model to a known "snapshot" time or to the beginning of the collaborative model execution. This would accommodate situations where internet performance is impacting the execution or model node issues require a "reboot".

Another area for future investigation can examine performance comparisons between shared memory and TCP/binary proxy throughput. This performance comparison can be gauged against an increasing number of nodes in a collaborative modeling architecture to determine breakpoints in the methodology[17].

17 This plug-in replacement for WebClient by Morten Nielsen adds support for GZIP compressed web requests – this reduces network traffic even further, apparently boosting load performance significantly. You won't really notice the difference on such a small data files as used in this sample, but as your JSON return values get larger, so will be the impact of using this library. SharpGIS.GZipWebClient

APPENDIX A: RCM THREAD LOGIC

The logic updates within the models is described below that supports each of the sync IDs identified in

```
using System.Linq;
using System.Web;
using System.Threading;
using System.Configuration;
using System.Web.Mvc;
using System.IO;
using java.lang;
using controller;
using rcm.Concrete;

namespace rcm.Concrete
{
    public class rcmThread
    {
        private int p;
        public java.lang.Object parItem;
        public double[] data = { 0, 0, 0, 0, 0, 0, 0, 0 };
        public bool initial = true;
        public static int cycleCounter = 0;
        public static int updateCounter = 0;
        public bool firstStepA = true;
        public bool firstStepC = true;

        SendToNode x = new SendToNode();
        StepNode s = new StepNode();
        GetFromNode g = new GetFromNode();
        controller.Controller c = new controller.Controller();
        RcmViewPost update = new RcmViewPost();

        // Zero-parameter constructor
        public rcmThread() { }

        // One-parameter constructor
        public rcmThread(int p) { this.p = p; }

        public void runRcm()
        {
            LogfileManager log = new LogfileManager();
            // appendLogfile method arguments: DateTime, String, String, String
            log.appendLogfile(DateTime.Now, "Started Execution Cycle", " - Node B
ThreatMotion Model", "");

            // "sync" is initialized to "stop" in Sync Controller; setting model state on
RCM View will stop
            // further communication from occuring with the DEVS models
            while ((rcm.Controllers.SyncController.sync != "stop") ||
rcm.Controllers.ContentController.mState != "Stop")
            {
                // Build schedule here for collaborative model execution
                // This collaborative model executes in two phases
```

```
                //      phase 1: Cycle 1 through cycle 15
                //      Phase 2: Repeating 13 cycle intervals
                if (rcm.Controllers.ContentController.mState == "Start") // Phase 1
                {
                    // Start position and motion set to "A1" in RcnGeneratorB
                    c.userGesture("STEP", parItem); // Step B; Initializes the system

System.Threading.Thread.Sleep(rcm.Controllers.ContentController.throttle); // Wait for
output
                    g.getNodeB();
                    if (rcm.Concrete.GetFromNode.outputB) // Trackdata output?
                    {
                        x.inputA(); // Couple B to A - ThreatMotion to Datalink

System.Threading.Thread.Sleep(rcm.Controllers.ContentController.throttle); // Allow RCN
to respond
                        s.stepA();
                        if (firstStepA)
                        {
                            System.Threading.Thread.Sleep(8000); // Allow node A to
initialize
                            firstStepA = false;
                        }

System.Threading.Thread.Sleep(rcm.Controllers.ContentController.throttle); // Allow RCN
to respond
                        s.stepA();
                        rcm.Controllers.ContentController.mState = "Continue";
                    }
                    rcm.Concrete.GetFromNode.outputB = false;
                    cycleCounter++;
                }
                else if (rcm.Controllers.ContentController.mState == "Continue") // Phase
2
                {

System.Threading.Thread.Sleep(rcm.Controllers.ContentController.throttle); // Wait for
output
                    g.getNodeA();
                    if (rcm.Concrete.GetFromNode.outputA)
                    {
                        x.inputC(); // Couple A to C - Datalink to UsvAlgorithm
                        c.userGesture("STEP", parItem); // Step B

System.Threading.Thread.Sleep(rcm.Controllers.ContentController.throttle); // Wait for
output
                        s.stepC();
                        if (firstStepC)
                        {
                            System.Threading.Thread.Sleep(5000); // Allow node C to
initialize
                            firstStepC = false;
                        }
                    }
```

```
System.Threading.Thread.Sleep(rcm.Controllers.ContentController.throttle); // Wait for
output
                    g.getNodeC();
                    if (rcm.Concrete.GetFromNode.outputC)
                    {
                        x.inputB(); // Couple C to B - UsvAlgorithm to DataCollector

System.Threading.Thread.Sleep(rcm.Controllers.ContentController.throttle); // Wait for
output
                        c.userGesture("STEP", parItem); // Step B
                        s.stepC();
                    }

System.Threading.Thread.Sleep(rcm.Controllers.ContentController.throttle); // Wait for
output
                    g.getNodeB();
                    if (rcm.Concrete.GetFromNode.outputB)
                    {
                        x.inputA(); // Couple B to A - ThreatMotion to Datalink

System.Threading.Thread.Sleep(rcm.Controllers.ContentController.throttle); // Wait for
output
                        s.stepA();
                        c.userGesture("STEP", parItem); // Step B
                        s.stepC();
                        s.stepA();
                    }
                    if (!(rcm.Concrete.GetFromNode.outputA ||
rcm.Concrete.GetFromNode.outputB || rcm.Concrete.GetFromNode.outputC))
                    {
                        c.userGesture("STEP", parItem); // Step B
                        s.stepC();
                    }
                    rcm.Concrete.GetFromNode.outputA = false;
                    rcm.Concrete.GetFromNode.outputB = false;
                    rcm.Concrete.GetFromNode.outputC = false;
                    cycleCounter++;
                    /*if (updateCounter > 19)
                    {
                        update.postData();
                        updateCounter = 0;
                    }
                    else
                        updateCounter++;
                     * */
                }

            }
            // appendLogfile method arguments: DateTime, String, String, String
            log.appendLogfile(DateTime.Now, "Completed Execution Cycle", " - Node B
ThreatMotion Model", "");
        }
    }

}
```

APPENDIX B: INVENTORY OF PROJECT SOFTWARE

The following table outlines the software products used to support code development and RCMS application execution and DEVS model execution in this project:

Table 4 RCMS Software Inventory

Software Item	Version	Description	Accessed from
Microsoft Visual Studio 2012	1.0.x	Integrated Development Environment utilizing ASP.NET 4.5 and C#	http://www.microsoft.com/visualstudio/eng#downloads+d-express-web
Microsoft SQL Server 2012 LocalDB Express	1.0.x	Structured Query Language database server to support user account and documentation metadata information	http://www.microsoft.com/en-us/download/detailsaspx?id=29062
JNBridgePro	V7.0	JNBridgePro creates the interoperability bridge by generating a set of proxies that expose the classes' APIs and manage the communications between the .NET and Java classes.	http://www.jnbridge.com/docs.htm
Eclipse	Juno	Multi-language software development environment used for Java model development.	http://www.eclipse.org/downloads/
JDK	7u21	Java development kit	http://www.oracle.com/technetwork/java/javase/downloads/index.html
DEVS-Suite	2.0.0	DEVS simulator that combines the capabilities of DEVSJAVA and DEVS Tracking Environment.	http://acims.asu.edu/software/devs-suite
Entity Framework	EF-5	ADO.NET open source object-relational mapping framework for the .NET Framework.	http://entityframework.codeplex.com/releases
Json.Net	5.0	JavaScript Object Notation (JSON) Framework for .NET	http://json.codeplex.com/

APPENDIX C: SOURCE CODE FOR DEVS MODELS

Source code for the RCN Node A DataLink model follows. It consists of four Java classes as listed below. Underlying class source code that supports the DEVS-Suite simulation are not shown but can be referenced through the ACIMS website (http://acims.asu.edu/); the DEVS-Suite software can be downloaded from the SourceForge website (http://sourceforge.net/projects/devs-suitesim/):

- RestfulCollavoborativeNode
- RcnGenerator
- RcnTransducer
- Datalink (Refer to the reference [16] for source code; no modifications)

```
package SimpArcMod;  // RestfulCollaborativeNode Class

/*
 *  Author     : Jerome B. Heiser
 *  Version    : DEVSJAVA 2.7
 *  Date       : 03-08-13
 */

import java.awt.*;
import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.modeling.ViewableComponent;
import view.modeling.ViewableDigraph;
import view.simView.*;

public class RestfulCollaborativeNode extends ViewableDigraph{

       public RestfulCollaborativeNode(){
              super("RestfulCollaborativeNode");

              ViewableAtomic RcnGenerator = new RcnGenerator("RcnGenerator",1.0);
              ViewableAtomic Datalink = new Datalink("Datalink",1.0);
              ViewableAtomic RcnTransducer = new
              RcnTransducer("RcnTransducer",18000.0);

              add(RcnTransducer);
              add(Datalink);
              add(RcnGenerator);

              addInport("rcn_in1");  // Sync signal input "string"
              addInport("rcn_in2");  // Content - message bag from facade
              addOutport("rcn_out1");

              addTestInput("rcn_in1",new entity("Run"));
```

```
            addTestInput("rcn_in1",new entity("Pause"));
            addTestInput("rcn_in1",new entity("Step"));
            addTestInput("rcn_in1",new entity("Stop"));
            addTestInput("rcn_in1",new entity("Abort"));

            /*
             * Coupling Requirements
             */
            addCoupling(this,"rcn_in1",RcnGenerator,"rcnG_in1");
            addCoupling(this,"rcn_in2",RcnGenerator,"rcnG_in2");

            addCoupling(RcnGenerator,"rcnG_out1",Datalink,"dl_in");
            addCoupling(Datalink,"dl_out",RcnTransducer,"rcnT_in1");

            addCoupling(RcnTransducer,"rcnT_out1",this,"rcn_out1");

            //    initialize();
            showState();
      }

    /**
     * Automatically generated by the SimView program.
     * Do not edit this manually, as such changes will get overwritten.
     */
    public void layoutForSimView()
    {
        preferredSize = new Dimension(958, 281);
        ((ViewableComponent)withName("RcnGenerator")).setPreferredLocation(new
Point(85, 39));
        ((ViewableComponent)withName("Datalink")).setPreferredLocation(new Point(303,
181));
        ((ViewableComponent)withName("RcnTransducer")).setPreferredLocation(new
Point(501, 42));
    }
}
```

```
package SimpArcMod;  // RcnGenerator Class

/*
 *  Author     : Jerome B. Heiser
 *  Version    : DEVSJAVA 2.7
 *  Date       : 03-23-13
 */

import java.lang.*;
import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.simView.*;

public class RcnGenerator extends ViewableAtomic{
```

```
protected double generation_interval;
protected int count;

// ************************************************************************
//                    Model Data Definition Area
// Defines data to support model interface data generation requirements
// This area is to be updated by the modeler to comply with output from
// the Generator to what is needed by the Model
public static double[] dataInput; // Surveillance threat track data
public static int arraySize;
protected double incoming_dataInput;
protected double[] temp;
// ************************************************************************

public RcnGenerator() {this("RcnGenerator", 1.0);}

public RcnGenerator(String name,double Generation_Interval){
       super(name);
       addInport("rcnG_in1"); // Sync signal input
       addInport("rcnG_in2"); // Content input
       addOutport("rcnG_out1");
       generation_interval = Generation_Interval ;
       addTestInput("rcnG_in1",new entity("Step"));
}

// ************************************************************************
// Defined per modeler configuration data
// ************************************************************************
public void initialize(){
       phase = "wait";
       sigma = generation_interval;
       count = 0;
       dataInput = new double[8]; // Must correspond with "arraySize"
       arraySize = 8;
       super.initialize();
}

// External transition function called by the RCN facade through the
// Controller.injectInputGesture method. Data passed in the message bag
// consists of the Sync signal the model input parameters as defined
// within the configuration data provided by the modeler
public void  deltext(double e,message x)
{
       Continue(e);
       if(phaseIs("active")){
              for (int i=0; i< x.getLength();i++) {
                     if (messageOnPort(x,"rcnG_in2",i)){
                            entity val = x.getValOnPort("rcn_in2",i);
                            arrayEnt incoming_dataInput = (arrayEnt)val;
                            temp = incoming_dataInput.getv();
                            dataInput = temp;
                            for (int j=0; j < arraySize; j++)
```

```
                                        System.out.println("dataInput" + j + ":  " +
                                        dataInput[j]);
                                holdIn("passive",0); // Force output immediately
                        }
                }
        }
        if(phaseIs("wait")){
                for (int i=0; i< x.getLength();i++)
                        if (messageOnPort(x,"rcnG_in1",i)) // Step signal
                                phase = "active"; // Wait for data input
        }
}

// Override for deltext function
// Access to inject method not attainable
// Access to message type structure not attainable
public static void altdeltext(double[] in)
{
        for (int i = 0; i < arraySize; i++){
                dataInput[i] = in[i];
        }
}

public void  deltint( )
{
        holdIn("wait",generation_interval);
}

public void deltcon(double e,message x)
{
   deltint();
   deltext(0,x);
}

public message  out( )
{
        message  m = new message();
        m.add(makeContent("rcnG_out1", new arrayEnt(dataInput)));
        return m;
}

public void showState(){ super.showState(); }

public String getTooltipText(){
        return
                        super.getTooltipText()
                        +"\n"+" int_arr_time: " + generation_interval
                        +"\n"+" count: " + count;
}
}
```

```java
package SimpArcMod;  // RcnTransducer Class

/*
 *  Author     : Jerome B. Heiser
 *  Version    : DEVSJAVA 2.7
 *  Date       : 03-23-13
 */

import java.util.HashMap;
import java.util.Map;
import GenCol.arrayEnt;
import GenCol.doubleEnt;
import GenCol.entity;
import model.modeling.content;
import model.modeling.message;
import view.modeling.ViewableAtomic;
import java.lang.Object;
import java.io.OutputStream;
import java.io.FileOutputStream;
import java.io.DataOutputStream;
import java.io.BufferedOutputStream;

public class RcnTransducer extends  ViewableAtomic{
	protected Map logfile, errorLog;
	protected double[] temp;
	protected double[] outputData;
	public static double[] nodeOutput = new double[] {0,0,0,0,0,0,0,0};
	static final String log = "logfile";
	static final String debug = "errorLog";
	protected DataOutputStream out, out1;
	protected double clock, observation_time;
	public static int arraySize;
	public static boolean newdata = false;


	public RcnTransducer(String  name,double Observation_time){
		super(name);
		addInport("rcnT_in1");
		addOutport("rcnT_out1");

		logfile = new HashMap();
		errorLog = new HashMap();

		observation_time = Observation_time;
		initialize();
	}

	public RcnTransducer() {this("RcnTransducer", 18000);}

	public void initialize(){
		phase = "start";
		sigma = observation_time;
```

```
		clock = 0;
		arraySize = 8;

		// Open data collection files: "logfile" and "errorLog"
		try { out = new DataOutputStream(new FileOutputStream(log)); }
		catch (Exception a) {System.err.println("Error opening logfile");}

		try { out1 = new DataOutputStream(new FileOutputStream(debug)); }
		catch (Exception a) {System.err.println("Error opening errorLog file");}

		super.initialize();
	}

	public void showState(){
		super.showState();
	}

	public void  deltext(double e,message  x){
		clock = clock + e;
		Continue(e);
		for(int i=0; i< x.size();i++){
			if(messageOnPort(x,"rcnT_in1",i)){
				// Collect the data input to the Transducer
				entity val = x.getValOnPort("rcnT_in1",i);
				arrayEnt inputData = (arrayEnt)val;
				temp = inputData.getv();
				outputData = temp;
				nodeOutput = temp; // Store data output for RCN
				newdata = true; // Set new data flag for RCN
				logfile.put(new doubleEnt(clock),temp);
				phase = "active";

				// Outputting data to file
				try
				{
					// out.writeDouble(temp[j]);
					out.writeChars(Double.toString(temp[0])+"    ");
					out.writeChars(Double.toString(temp[1])+"    ");
					out.writeChars("\n");
				}
				catch (Exception a)
				{
					System.err.println("Error writing to file");
					errorLog.put(new doubleEnt(clock), "Error writing
to file");
				}
			}
		  }
		show_state();
	}

	public void  deltint(){
		clock = clock + sigma;
		passivate();
```

```
            show_state();
        }

        // Determine if final output collection is necessary
        public  message    out( ){
            message  m = new message();
            content  con1 = makeContent("rcnT_out1",new arrayEnt(outputData));;
            m.add(con1);
            return m;
        }

        // Psuedo facade to move data to RCN facade
        // Method called through JNBridge proxy
        public static double[] altout(){
            newdata = false; // Reset new data flag for RCN
            return nodeOutput;
        }

        public void  show_state(){

            System.out.println("state of  "  +  name  +  ": " );
            System.out.println("phase, sigma : "
       + phase  +  " "  +  sigma  +  " "   );
        }
}
```

Source code for the RCN Node C UsvAlgorithm model follows. It consists of four Java classes as listed below. Underlying class source code that supports the DEVS-Suite simulation are not shown but can be referenced through the ACIMS website (http://acims.asu.edu/); the DEVS-Suite software can be downloaded from the SourceForge website (http://sourceforge.net/projects/devs-suitesim/):

- RestfulCollavoborativeNodeC
- RcnGenerator (Same code as used for Datalink model; refer to above source code)
- RcnTransducerC
- UsvAlgorithm (Refer to the reference [16] for source code; no modifications)

```
package SimpArcMod;  // RestfulCollaborativeNodeC Class

/*
 *  Author     : Jerome B. Heiser
 *  Version    : DEVSJAVA 2.7
 *  Date       : 03-25-13
 */
```

```
import java.awt.*;
import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.modeling.ViewableComponent;
import view.modeling.ViewableDigraph;
import view.simView.*;

public class RestfulCollaborativeNodeC extends ViewableDigraph{

	public RestfulCollaborativeNodeC(){
		super("RestfulCollaborativeNodeC");

		ViewableAtomic RcnGenerator = new RcnGenerator("RcnGenerator",1.0);
		ViewableAtomic UsvAlgorithm = new UsvAlgorithm("UsvAlgorithm",1.0, "A");
		ViewableAtomic RcnTransducerC = new
RcnTransducerC("RcnTransducerC",18000.0);

		add(RcnTransducerC);
		add(UsvAlgorithm);
		add(RcnGenerator);

		addInport("rcn_in1");  // Sync signal input "string"
		addInport("rcn_in2");  // Content - message bag from facade
		addOutport("rcn_out1");

		addTestInput("rcn_in1",new entity("Run"));
		addTestInput("rcn_in1",new entity("Pause"));
		addTestInput("rcn_in1",new entity("Step"));
		addTestInput("rcn_in1",new entity("Stop"));
		addTestInput("rcn_in1",new entity("Abort"));

		/*
		 * Coupling Requirements
		 */
		addCoupling(this,"rcn_in1",RcnGenerator,"rcnG_in1");
		addCoupling(this,"rcn_in2",RcnGenerator,"rcnG_in2");

		addCoupling(RcnGenerator,"rcnG_out1",UsvAlgorithm,"ua_in1");
		addCoupling(UsvAlgorithm,"ua_out1",RcnTransducerC,"rcnT_in1");
		addCoupling(UsvAlgorithm,"ua_out2",RcnTransducerC,"rcnT_in2");

		addCoupling(RcnTransducerC,"rcnT_out1",this,"rcn_out1");

		//    initialize();
		showState();

	}

    /**
     * Automatically generated by the SimView program.
     * Do not edit this manually, as such changes will get overwritten.
     */
```

```
    public void layoutForSimView()
    {
        preferredSize = new Dimension(624, 281);
        ((ViewableComponent)withName("RcnTransducerC")).setPreferredLocation(new
Point(246, 116));
        ((ViewableComponent)withName("UsvAlgorithm")).setPreferredLocation(new
Point(106, 213));
        ((ViewableComponent)withName("RcnGenerator")).setPreferredLocation(new
Point(-23, 82));
    }
}
```

```
package SimpArcMod;  // RcnTransducerC Class

/*
 *  Author     : Jerome B. Heiser
 *  Version    : DEVSJAVA 2.7
 *  Date       : 03-25-13
 */

import java.util.HashMap;
import java.util.Map;
import GenCol.arrayEnt;
import GenCol.doubleEnt;
import GenCol.entity;
import model.modeling.content;
import model.modeling.message;
import view.modeling.ViewableAtomic;
import java.lang.Object;
import java.io.OutputStream;
import java.io.FileOutputStream;
import java.io.DataOutputStream;
import java.io.BufferedOutputStream;

public class RcnTransducerC extends  ViewableAtomic{
       protected Map logfile, errorLog;
       protected double[] temp, outputData;
       static double[] nodeOutput = new double[] {0,0,0,0,0,0,0,0};
       static final String log = "logfile";
       static final String debug = "errorLog";
       protected DataOutputStream out, out1;
       protected double clock, observation_time;
       protected int arraySize;
       public static boolean newdata = false;

       public RcnTransducerC(String  name,double Observation_time){
              super(name);
              addInport("rcnT_in1");
```

```
        addInport("rcnT_in2");
        addOutport("rcnT_out1");

        logfile = new HashMap();
        errorLog = new HashMap();

        observation_time = Observation_time;
        initialize();
}

public RcnTransducerC() {this("RcnTransducerC", 18000);}

public void initialize(){
        phase = "start";
        sigma = observation_time;
        clock = 0;
        arraySize = 9;

        // Open data collection files: "logfile" and "errorLog"
        try { out = new DataOutputStream(new FileOutputStream(log)); }
        catch (Exception a) {System.err.println("Error opening logfile");}

        try { out1 = new DataOutputStream(new FileOutputStream(debug)); }
        catch (Exception a) {System.err.println("Error opening errorLog file");}

        super.initialize();
}

public void showState(){
        super.showState();
}

public void  deltext(double e,message  x){
        clock = clock + e;
        Continue(e);
        for(int i=0; i< x.size();i++){
                if(messageOnPort(x,"rcnT_in1",i)){
                        // Collect the data input to the Transducer
                        entity val = x.getValOnPort("rcnT_in1",i);
                        arrayEnt inputData = (arrayEnt)val;
                        temp = inputData.getv();
                        outputData = temp;
                        nodeOutput = temp; // Data output for RCN
                        newdata = true; // Set new data flag for RCN
                        logfile.put(new doubleEnt(clock),temp);
                        phase = "active";


                        // Outputting data to file
                        try
                        {
                                // out.writeDouble(temp[j]);
                                out.writeChars(Double.toString(temp[0])+"    ");
                                out.writeChars(Double.toString(temp[1])+"    ");
```

```
                                        out.writeChars("\n");
                                }
                                catch (Exception a)
                                {
                                        System.err.println("Error writing to file");
                                        errorLog.put(new doubleEnt(clock), "Error writing
to file");
                                }
                        }
                  }
                show_state();
        }

        public void  deltint(){
                clock = clock + sigma;
                passivate();
                show_state();
        }

        // Determine if final output collection is necessary
        public  message    out( ){
                message  m = new message();
                content  con1 = makeContent("rcnT_out1",new arrayEnt(outputData));;
                m.add(con1);
                return m;
        }

        // Psuedo facade goes here to move data to RCN facade
        public double[] altout(){
                newdata = false; // Reset new data flag for RCN
                return nodeOutput;
        }

        public void  show_state(){

                System.out.println("state of  "  +  name  +  ": " );
                System.out.println("phase, sigma : "
           + phase  +  " "  +  sigma  +  " "   );
        }
}
```

Source code for the RCM Node B ThreatMotion and DataCollector models follow. It consists of four Java classes as listed below. Underlying class source code that supports the DEVS-Suite simulation are not shown but can be referenced through the ACIMS website (http://acims.asu.edu/); the DEVS-Suite software can be downloaded from the SourceForge website (http://sourceforge.net/projects/devs-suitesim/):

- RestfulCollavoborativeNodeB
- RcnGeneratorB
- RcnTransducer (Same code as used for Datalink model; refer to above source code)
- ThreatMotion and DataCollector (Refer to the reference [16] for source code; no modifications)

```
package SimpArcMod;  // RestfulCollaborativeNodeB Class

/*
 *  Author     : Jerome B. Heiser
 *  Version    : DEVSJAVA 2.7
 *  Date       : 04-01-13
 */

import java.awt.*;
import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.modeling.ViewableComponent;
import view.modeling.ViewableDigraph;
import view.simView.*;

public class RestfulCollaborativeNodeB extends ViewableDigraph{

       public RestfulCollaborativeNodeB(){
              super("RestfulCollaborativeNodeB");

              ViewableAtomic RcnGeneratorB = new RcnGeneratorB("RcnGeneratorB",1.0);
              ViewableAtomic ThreatMotion = new ThreatMotion("ThreatMotion",1.0);
            ViewableAtomic DataCollector = new DataCollector("DataCollector",18000.0);
              ViewableAtomic RcnTransducer = new
RcnTransducer("RcnTransducer",18000.0);

              add(RcnTransducer);
              add(DataCollector);
            add(ThreatMotion);
              add(RcnGeneratorB);

              addInport("rcn_in1");  // Sync signal input "string"
              addInport("rcn_in2");  // Content - message bag from facade
              addOutport("rcn_out1");
              addOutport("rcn_out2");
              addOutport("rcn_out3");

              addTestInput("rcn_in1",new entity("Run"));
              addTestInput("rcn_in1",new entity("Pause"));
              addTestInput("rcn_in1",new entity("Step"));
              addTestInput("rcn_in1",new entity("Stop"));
              addTestInput("rcn_in1",new entity("Abort"));
```

```
            /*
             * Coupling Requirements
             */
            addCoupling(this,"rcn_in1",RcnGeneratorB,"rcnG_in1");
            addCoupling(this,"rcn_in2",RcnGeneratorB,"rcnG_in2");

            addCoupling(RcnGeneratorB,"rcnG_out1",ThreatMotion,"tm_in1");
            addCoupling(RcnGeneratorB,"rcnG_out2",DataCollector,"dc_in2");
            addCoupling(RcnGeneratorB,"rcnG_out3",DataCollector,"dc_in3");
            addCoupling(ThreatMotion,"tm_out1",DataCollector,"dc_in1");
            addCoupling(ThreatMotion,"tm_out2",RcnTransducer,"rcnT_in1");

            addCoupling(RcnTransducer,"rcnT_out1",this,"rcn_out1");
            addCoupling(DataCollector,"dc_out1",this,"rcn_out2");
            addCoupling(DataCollector,"dc_out2",this,"rcn_out3");

            //    initialize();
            showState();

      }

    /**
     * Automatically generated by the SimView program.
     * Do not edit this manually, as such changes will get overwritten.
     */
    public void layoutForSimView()
    {
        preferredSize = new Dimension(796, 281);
        ((ViewableComponent)withName("RcnGeneratorB")).setPreferredLocation(new
Point(10, 56));
        ((ViewableComponent)withName("DataCollector")).setPreferredLocation(new
Point(371, 193));
        ((ViewableComponent)withName("ThreatMotion")).setPreferredLocation(new
Point(140, 195));
        ((ViewableComponent)withName("RcnTransducer")).setPreferredLocation(new
Point(352, 66));
    }
}
```

```
package SimpArcMod;  // RcnGeneratorB Class

/*
 *  Author     : Jerome B. Heiser
 *  Version    : DEVSJAVA 2.7
 *  Date       : 04-01-13
 */
```

```
import java.lang.*;
import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.simView.*;

public class RcnGeneratorB extends ViewableAtomic{

	protected double generation_interval;
	protected int count;

	// ***************************************************************
	//                    Model Data Definition Area
	// Defines data to support model interface data generation requirements
	// This area is to be updated by the modeler to comply with output from
	// the Generator to what is needed by the Model
	public static double[] dataInput; // Surveillance threat track data
	public static int arraySize;
	protected double incoming_dataInput;
    protected double[] temp;
    // ***************************************************************

	public RcnGeneratorB() {this("RcnGeneratorB", 1.0);}

	public RcnGeneratorB(String name,double Generation_Interval){
		super(name);
		addInport("rcnG_in1"); // Sync signal input
		addInport("rcnG_in2"); // Content input
		addOutport("rcnG_out1"); // Initialization value
		addOutport("rcnG_out2"); // USV Kinematics data
		addOutport("rcnG_out3"); // TTI and Error data
		generation_interval = Generation_Interval ;
	}

	// ***************************************************************
	// Defined per modeler configuration data
	// ***************************************************************
	public void initialize(){
		phase = "wait";
		sigma = generation_interval;
		count = 0;
		dataInput = new double[8]; // Must correspond with "arraySize"
		arraySize = 8;
		super.initialize();
	}

	// External transition function called by the RCN facade through the
	// Controller.injectInputGesture method. Data passed in the message bag
	// consists of the Sync signal the model input parameters as defined
	// within the configuration data provided by the modeler
	public void  deltext(double e,message x)
	{
		Continue(e);
```

```
		if(phaseIs("active")){
			for (int i=0; i< x.getLength();i++) {
				if (messageOnPort(x,"rcnG_in2",i)){
					entity val = x.getValOnPort("rcn_in2",i);
					arrayEnt incoming_dataInput = (arrayEnt)val;
					temp = incoming_dataInput.getv();
					dataInput = temp;
					for (int j=0; j < arraySize; j++)
						System.out.println("dataInput" + j + ":  " +
dataInput[j]);
					holdIn("passive",0); // Force output immediately
				}
			}
		}
		if(phaseIs("wait")){
			for (int i=0; i< x.getLength();i++)
				if (messageOnPort(x,"rcnG_in1",i)) // Step signal
					phase = "active"; // Wait for data input
		}
	}

	// Override for deltext function
	// Access to inject method not attainable
	// Access to message type structure not attainable
	public static void altdeltext(double[] in)
	{
		for (int i = 0; i < arraySize; i++){
			dataInput[i] = in[i];
		}
	}

	public void  deltint( )
	{
		holdIn("wait",generation_interval);
	}

	public void deltcon(double e,message x)
	{
	   deltint();
	   deltext(0,x);
	}

	public message  out( )
	{
		message  m = new message();
		// Initialize ThreatMotion to straight line motion
		// After initial output further output messages will have no affect
		m.add(makeContent("rcnG_out1", new entity("A1")));
		return m;
	}

	public void showState(){
		super.showState();
	}
```

```
	public String getTooltipText(){
		return
				super.getTooltipText()
				+"\n"+" int_arr_time: " + generation_interval
				+"\n"+" count: " + count;
	}

}
```

APPENDIX D: INSTRUCTION SET FOR GENERATING .NET TO JAVA PROXY "DLL"

(1) If Java files have been modified, updated, added or deleted, the DEVS-Suite_jbh.jar will have to be recreated. Follow the indented steps below. Otherwise proceed to step 2.
    a. Right-click on the DEVS-Suite package and select Export … ; in the Export window select Jar File;
    b. In the Jar Export window select DEVS-Suite and uncheck the DEVS-Suite_jbh.jar file and all the data files (e.g. error.log, logfile.txt, etc.);
    c. Under "Select to export destination", browse to the classpath location: C:\Users\JBH630\workspace2\DEVS-Suite and enter the DEVS-Suite_jbh.jar filename to the path in the Jar File entry field.
    d. Select Finish and after the export is complete ensure the file DEVS-Suite_jbh.jar has been updated in the destination folder.
(2) Open JNBridgePro proxy generation tool and select OK
(3) On the Launch JNBProxy window, select the ***Create new .Net -> Java project*** radio button, and then select OK
(4) On the JNBProxy GUI, select ***Project -> Java Options*** from the menu
(5) On the Enter Java Options window, make the following selections:
    a. Under Choose the communications mechanism, select the ***Binary/TCP*** radio button; set the Remote host to ***localhost***; and set the Remote port to ***8085***
    b. Ensure the Start Java automatically check-box is checked
    c. Under Locate the java.exe file browse to and select the path C:\Program Files\Java\jdk1.7.0_15\bin\java.exe
    d. Under Locate the jnbcore.jar file browse to and select the path C:\Program Files (x86)\JNBridge\JNBridgePro v7.0\jnbcore\jnbcore.jar
    e. Under Locate the bcel.jar file browse to and select the path C:\Program Files (x86)\JNBridge\JNBridgePro v7.0\jnbcore\bcel-5.1-jnbridge.jar
    f. Ensure the Generate J2SE 5.0-targeted check-box is un-checked
    g. Select OK
(6) On the JNBProxy GUI, select ***Project -> Edit Classpath*** from the menu
(7) On the Edit Class Path wind select the Add button
(8) On the New Classpath Element, select the DEVS-Suite_jbh.jar file browsing to the path C:\Users\JBH630\workspace2\DEVS-Suite\DEVS-Suite_jbh.jar and then selecting OK
(9) On the JNBProxy GUI, select ***Project -> Add Classes from JAR file*** from the menu
(10) Select the DEVS-Suite_jbh.jar file by browsing to path C:\Users\JBH630\workspace2\DEVS-Suite
(11) On the JNBProxy GUI, select, in the Environment pane, check the following for which a proxy is to be generated:
    a. controller package
    b. GenCol package
    c. Java.lang package
    d. Under SimArcMod package

i. RcnGenerator
ii. RcnGeneratorB
iii. RcnTransducer
iv. RcnTransducerC
v. RestfulCollaborativeNode
vi. RestfulCollaborativeNodeA
vii. RestfulCollaborativeNodeB

(12) On the JNBProxy GUI, select the ***Add+*** button

(13) When the process completes on the JNBProxy GUI, select ***Project -> Build …*** from the menu

(14) On the Save generated proxies window, browse to path C:\Users\JBH630\Documents\Visual Studio 2012\Projects\rcn\rcn\bin and then enter the Filename MyJavaProxies

(15)Ensure MyJavaProxies is present at location C:\Users\JBH630\Documents\Visual Studio 2012\Projects\rcn\rcn\bin

APPENDIX E: .NET SIDE JNBRIDGE CONFIGURATIONS

JNBridge .Net Side Configuration Settings in Web.config

```
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" />
    <!-- JNBrdige configuration setting -->
    <sectionGroup name="jnbridge">
      <section name="licenseLocation"
type="System.Configuration.SingleTagSectionHandler"/>
      <section name="dotNetToJavaConfig"
          type="System.Configuration.SingleTagSectionHandler, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
      <section name="javaToDotNetConfig"
          type="System.Configuration.SingleTagSectionHandler, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
      <section name="tcpNoDelay"
          type="System.Configuration.SingleTagSectionHandler, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
      <section name="javaSideDeclarations"
          type="System.Configuration.NameValueSectionHandler, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
    </sectionGroup>

<jnbridge>
    <dotNetToJavaConfig scheme="jtcp"
              host="localhost"
              port="8085" />
    <!-- For most types of licenses, the license is encapsulated in a license file, which
is a text file whose suffix is .lic. -->
    <licenseLocation directory ="C:\Program Files (x86)\JNBridge\JNBridgePro v7.0" />
  </jnbridge
```

RCN Project JNBridge Refereneces and License Manager Files

```
Add to references:  JNBShare.dll and MyJavaProxies.dll
Add “existing items” to project: rlm932_x64.dll and rlm932_x86.dll [set the “Copy to
Output Directory” property to “Copy always”] Add (or ensure) the license file ending in
“*.lic” in folder "C:\Program Files (x86)\JNBridge\JNBridgePro v7.0"
```

APPENDIX F: JNBRIDGE JAVA SIDE CONFIGURATION CODE DEVELOPMENT

GenerateProperties.java

```
package SimpArcMod;  // GenerateProperties.java

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Properties;

public class GenerateProperties {

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		try {
			Properties properties = new Properties();
			properties.setProperty("javaSide.serverType", "tcp");
			properties.setProperty("javaSide.port", "8085");

			File file = new File("myJnbProp.properties");
			FileOutputStream fileOut = new FileOutputStream(file);
			properties.store(fileOut, ".Net -> Java Side Properties");
			fileOut.close();
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException ex) {
			ex.printStackTrace();
		}
	}
}
```

myJnbProp.properties

C:\Users\JBH630\workspace2\DEVS-Suite\myJnbProp.properties
#.NET -> JAVA SIDE PROPERTIES
#SAT MAR 16 11:41:52 EDT 2013
JAVASIDE.PORT=8085
JAVASIDE.SERVERTYPE=TCP

APPENDIX G: JAVA THREAD CODE

The Java thread code provides the means for the JBNMain to setup the selected TCP communication over the socket connection on port 80; these properties were set in the myJnbProp properties file. It requires the main Controller for DEVS Suite to be also started.

```
package SimpArcMod;  // MyJavaThread.java

import com.jnbridge.jnbcore.server.ServerException;

public class MyJavaThreads {


	public static void main(String[] args) {
		// JNBMain start thread
		Thread thread1 = new Thread() {
			public void run() {
				try {

	com.jnbridge.jnbcore.JNBMain.start("C:\\Users\\JBH630\\workspace2\\DEVS-
Suite\\myJnbProp.properties");
				} catch (ServerException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
			}
		};


		// DEVS-Suite Controller start thread
		Thread thread2 = new Thread() {
			public void run() {
				controller.Controller.main(null);
			}
		};

		thread1.start();
		thread2.start();
	}
}
```

APPENDIX H: RCN THREAD AND FAÇADE

The MyApplication thread shown here starts the RCN thread to manage data input to the DEVS model and model time advance through the step function. A similar thread was developed for the RCM node.

MyApplication Thread:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Threading;

namespace rcn.Concrete
{
    public class MyApplications
    {
            public static Semaphore cell;

            // Main function to create rcn thread and DEVS thread
            public void myProcesses()
            {
                // Create a semaphore for a resource in case it is needed
                cell = new Semaphore(0, 1);

                // Instantiate thread
                rcnThread t0 = new rcnThread(0);

                // Create the thread
                Thread rcnApp = new Thread(new ThreadStart(t0.runRcn));

                // Start thread
                rcnApp.Start();

                // Set semaphore to max value
                cell.Release(1);

                // Main waits for the threads to complete
                // rcnApp.Join();
                // The Join is carried out in rcnThread class
            }
    }
}
```

RCN Façade: rcnThread class instantiated in MyApplication

The RCN Façade acts as the go between the C# side and the Java side. It relies on the proxies in MyJavaProxies.dll to provide the connectivity.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Threading;
using System.Configuration;
using System.Web.Mvc;
using rcn.Models;
using System.IO;
using java.lang;
using controller;

namespace rcn.Concrete
{
    class rcnThread
    {
        private int p;
        public java.lang.Object parItem;
        public double[] data = {0,0,0,0,0,0,0,0};
        int arraySize = 8;
        controller.Controller c = new controller.Controller();

        // Zero-parameter constructor
        public rcnThread() { }

        // One-parameter constructor
        public rcnThread(int p) { this.p = p; }

        public void runRcn()
        {

            LogfileManager log = new LogfileManager();
            // appendLogfile method arguments: DateTime, String, String, String
            log.appendLogfile(DateTime.Now, "Started Execution Cycle", " - Node C
DataLink Model", "");

            // This thread will run continuously until actioned to stop
            while (rcn.Controllers.SyncController.sync != "stop")
            {
                // Step model through next time advance through Sync signal
                if (rcn.Controllers.SyncController.sync == "step")
                {
                    rcn.Controllers.SyncController.sync = "pause";
                    try
                    {
                        // Step the simulation
                        c.userGesture("STEP", parItem);
                    }
                    catch (TypeInitializationException ex)
                    {
```

```
                    System.Exception e = ex.InnerException;
                    while (e is TypeInitializationException)
                    {
                        e = e.InnerException;
                    }
                        StreamWriter sw =
                        File.AppendText(Path.Combine(Environment.GetFolderPath(Environ
                        ment.SpecialFolder.Personal), @"error.log"));
                    sw.WriteLine(e.Message + ": " + e.GetType().ToString() +
                    DateTime.Now.ToString());
                    sw.Close();
                }
            }

            // Check for input data from the RCM
            // The Content controller Input action sets the "content" variable
            if (rcn.Controllers.ContentController.content == "input")
            {
                try
                {
                        for (int i = 0; i < arraySize; i++) data[i] =
                        rcn.Controllers.ContentController.inputData[i];
                    SimpArcMod.RcnGenerator.altdeltext(data);
                    rcn.Controllers.ContentController.content = "null";
                }
                catch (System.Exception e)
                {
                        StreamWriter sw =
                        File.AppendText(Path.Combine(Environment.GetFolderPath(Environ
                        ment.SpecialFolder.Personal), @"error.log"));
                    sw.WriteLine(e.Message + ": " + e.GetType().ToString() +
                    DateTime.Now.ToString());
                    sw.Close();
                }
            }
        }
        // appendLogfile method arguments: DateTime, String, String, String
          log.appendLogfile(DateTime.Now, "Completed Execution Cycle", " - Node C
          DataLink Model", "");
    }
}
```

ACRONYMS

ACIMS	Arizona Center for Integrative Modeling and Simulation
API	Application Program Interface
ASP	Active Server Pages
CD++	
CLR	Common Language Runtime
CENTRA^2	Continuously Engaged Net-Centric Reconnaissance Autonomous Aerostat
DEVS	Discrete Event Systems Specification
DLL	Dynamic Link Library
EF	Entity Framework
GUI	Graphical User Interface
HATEOAS	Hypermedia As The Engine of Application State
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISO	Input/Step/Output
JNI	Java Native Interface
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MFVC	Model-Façade-View-Controller
MVC	Model-View-Controller
.NET	Not an acronym; stands for Microsoft's framework for writing web-enabled applications
ORM	Object Relational Mapping
PDF	Portable Document Format (Adobe Acrobat)
POCO	Plain Old CLR Object
RCM	RESTful Collaborative Model
RCMS	RESTful Collaborative Modeling System
RCN	RESTful Collaborative Node
REST	Representational State Transfer
RISE	RESTful Interoperability Simulation Environment

SME	Subject Matter Expert
SOA	Service Oriented Architecture
SQL	Structured Query Language
TTI	Time To Interdiction
URI	Universal Resource Identifier
USV	Unmanned Surface Vehicle
Vlab	Virtual Lab (Arizonia State University)
XML	Extensible Markup Language

REFERENCES

[1] Al-Zoubi, K., Wainer, G, "Performing Distributed Simulation with RESTful Web Services", Proceedings of the 2009 Winter Simulation Conference

[2] Li, y., SHen, J., Huang, Y., Shen, W., GHenniwa, H., Xu, Y., "Multi-Model Driven Collaborative Development Platform for Service-Oriented e-Business Systems", July 2008

[3] Xie, H., Boukerche, A., Zhang, M., "A Novel Message-Oriented and SOA Based Real-Time Modeling and Simulation Framework for Peer-to-Peer Systems", Proceedings of the 2009 Winter Simulation Conference

[4] Sarjoughian, H., Zeigler, B., Park, S., "Collaborative Distributed Network System: A Lightweight Middleware Supporting Collaborative DEVS Modeling", Future Generation Computer Systems 17 (2000) 89-105

[5] Al-Zoubi, K., Wainer, G,, "Distributed Simulation Using RESTful Interoperability Simulation Environment (RISE) Middleware",

[6] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Dissertation, University of California, Irvine, 2000

[7] Burke, B., "RESTful Java with JAX-RS", ISBN: 0596158041

[8] Hazlewood, L., "Designing a Beautiful REST + JSON API", http://www.youtube.com/watch?v=5WXYw4J4QOU

[9] Freeman, A., "Pro ASP.NET MVC 4", 4th Edition, Apress, 2012

[10] Al-Zoubi, K., Wainer, G., "Distributed Simulation Using RESTful Interoperability Simulation Environment (RISE) Middleware", Chap. 6, pp. 129-157, Intelligence-Based Systems Engineering, Springer-Verlag Berlin Heidelberg, 2011

[11] "Securing RESTful Web Services", http://www.layer7tech.com/solutions/securing-restful-web-services?gclid=CKfL76eN_7MCFWrZQgodNzMAaQ

[12] Al-Zoubi, K., Wainer, G. "Performing Distributed Simulation with RESTful Web-Services Approach". Proceedings of the Winter Simulation Conference (WSC 2009), Austin, TX, pp. 1323–1334 (2009)

[13] Al-Zoubi, K., Wainer, G., "Using REST Web Services Architecture for Distributed Simulation". Proceedings of Principles of Advanced and Distributed Simulation (PADS 2009), Lake Placid, New York, USA, pp. 114–121 (2009)

[14] Scott Allen, K., "HTTP: What Every Web Developer Should Know About HTTP", 2012, OdeToCode LLC

[15] Sarjoughian, H., "Component-based Simulators: DEVS-Suite Concepts, Techniques, and Operations", May 12,2010, Arizona Center for Integrative Modeling & Simulation

[16] Heiser, J., "Modeling Interdiction Methods", May 2009, CSE561 Modeling and Simulation course project

[17] Zeigler, B., Sarjoughian, H., "Introduction to DEVS Modeling and Simulation with Java: Developing Component-Based Simulation Models", January 2005, ACIMS, Tempe, AZ

[18] Kim, S., Sarjoughian, H., Elamvazhuthi, V., "DEVS-Suite: A Simulator Supporting Visual Experimentation Design and Behavior Monitoring", DEVS Integrative M&S Symposium, Proceedings of the Spring Simulation Conference, March 2009, San Diego, CA.

[19] Artificial Intelligence and Simulation Research Group, “DEVS-Java Reference Guide”, 30 September 1997, University of Arizona
[20] Heiser, J., “Exploring CENTRA^2 Command and Control Model Based Design”, May 2012, CSE522 Real Time Embedded Systems Design course project
[21] Nutaro, J., Sarjoughian, H., Zeigler, B., “Collaborative DEVS Modeler Users-Guide”, 16 August 1999, AI & Simulation Research Group, Electrical & Computer Engineering Department, University of Arizona
[22] Lerman, J., Miller, R., “Programming Entity Framework: Code First”, O’Reilly, Copyright 2012, ISBN: 978-1-449-31294-7
[23] Lerman, J., Miller, R., “Programming Entity Framework: DbContext”, O’Reilly, Copyright 2012, ISBN: 978-1-449-31296-1
[24] JNBridgePro™ Users’ Guide v7.0, JNBridge LLC, Copyright 2001-2013
[25] Mittal, S., Risco-Martín, J.L., Zeigler, B.P., “DEVSML- Automating DEVS Execution Over SOA Towards Transparent Simulators“, DEVS Symposium, Spring Simulation Multiconference, Norfork, Virginia, pp. 287-295, March
[26] Wainer, G., K. Al-Zoubi, O. Dalle, D. Hill, J. Martín, S. Mittal, H.S. Sarjoughian, L. Touraille, M. Traoré, B. Zeigler, 2011, “Standardizing DEVS Simulation Middleware”, Discrete Event Simulation and Modeling, G. Wainer and P. Mosterman, Editors, ISBN 459-493, 978-1420072334, CRC Press
[27] ACIMS software site: http://www.acims.arizona.edu/SOFTWARE/software.shtml
[28] Sarjoughian, H.S., Ziegler, B.P., “DEVS and HLA: Complimentary Paradigms for M&S”, Transactions of the SCS, (17), 4, pp 187-197, 2000
[29] Cho, Y., Zeigler, B.P., Sarjoughian, H.S., “Design and Implementation of Distributed Real-Time DEVS/CORBA”, IEEE Sys. Man Cyber Conference, Tucson, Oct 2001
[30] Wainer, G., Giambiase, N., “Timed Cell-DEVS: Modeling and Simulation of Cell-Spaces”, Invited paper for book Discrete Event Modeling & Simulation: Enabling Future Technologies, Springer-Verlag 2001
[31] Zhang, M., Zeigler, B.P., Hammonds, P., “DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies”, ITEA Journal, July 2005