USING THE COALITION BATTLE MANAGEMENT LANGUAGE STANDARD FOR INTERACTING WITH THE RESTFUL INTEROPERABILITY SIMULATION ENVIRONMENT

by

Elizabeth Hosang, P. Eng.

A thesis submitted to

The Faculty of Graduate Studies and Research

In partial fulfillment of the

requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario

Canada

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

acceptance of the thesis

Using the Coalition Battle Management Language Standard for Interacting with the RESTful Interoperability Simulation Environment

Submitted by Elizabeth Hosang

In partial fulfillment of the requirements of the

Degree of Master of Applied Science

_____

Thesis Supervisor

Dr. Gabriel Wainer

_____

Chair, Department of Systems and Computer Engineering

Dr. Roshdy Hafez

Carleton University

2014

Abstract

Computer simulation is used for planning and training for cases where the real-life scenario is expensive, rare or dangerous. Interoperability techniques allow diverse models built using different technologies on different hardware platforms to interact to create a larger, more complex synthetic environment. The Coalition Battle Management Language (C-BML) standard provides a set of definitions that can be used to communicate a commander's intent. The Military Scenario Definition Language (MSDL) uses the same building blocks to construct an initial definition of a scenario that is to be executed. The Discrete Event Simulation (DEVS) methodology provides a technique for modeling systems that react to external input in the form of events. The RESTful Interoperability Simulation Environment (RISE) provides a web-enabled platform that hosts the execution of DEVS models. This thesis proposes an architecture which adds the capability of using a structured scenario definition language file based on MSDL to initialize a DEVS model, and to provide a structured message based on C-BML as the initial input to a DEVS model. This architecture is validated through the execution of a scenario where civilian emergency services respond to an emergency.

Acknowledgements

This work is dedicated to my daughter Michelle for her patience over the past four years, my parents for their help, and my extended family and friends for their support. I also would like to thank Dr. Gabriel Wainer for his advice.

Table of Contents

Table of Figures

Table of Tables

Acronyms

The following acronyms were used in the document.

Acronym	Definition
API	Application Programming Interface
APP-6A	Allied Procedural Publication 6A
AT/FP	Anti-Terrorism/Force Protection
BML	Battle Management Language
BOM	Base Object Model
C2	Command and Control
C3	Command, Control and Communications
C4I	Command, Control, Communications, Computing and Intelligence
C-BML	Coalition Battle Management Language
CGF	Computer Generated Forces
DARPA	Defense Advanced Research Projects Agency
DEVS	Discrete EVent System
DIS	Distributed Interactive Simulation
FOM	Federation Object Model
G-DEVS	Generalized Discrete Event System
GMU	George Mason University
HLA	High Level Architecture
HTTP	Hypertext Transfer Protocol
JC3IEDM	Joint Command, Control and Communication Information Exchange Data Model
MIP	Multilateral Interoperability Programme
MSDL	Military Scenario Definition Language
MOOTW	Military Operations Other Than War
NATO	North Atlantic Treaty Organisation
NKN	Not Known
NMSG	NATO Modelling and Simulation Group

NOS	Not Otherwise Specified
OID	Object Identification
ORBAT	ORder of BATtle
PDU	Protocol Data Unit
REST	Representational State Transfer. Services using REST are described as RESTful.
RISE	RESTful Interoperability Simulation Environment
RPR FOM	Real-Time Platform Reference Federation Object Model
RTI	Run Time Infrastructure
SBML	Scripted Battle Management Language
SISO	Simulation Interoperability Standards Organization
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
URI	Uniform Resource Identifier
USV	Unmanned Surface Vehicle
UUV	Unmanned Undersea Vehicle
XML	Extensible Markup Language

1 Introduction

1.1 Motivation and Goals

Simulation software is being used more and more to allow planners to study situations that are too complicated or expensive to execute in real life. Militaries in particular use simulation for mission rehearsal in order to evaluate possible courses of action prior to deployment. Modeling tools such as Computer Generated Forces (CGF) systems allow the simulation of deployment of troops and equipment without putting personnel at risk or incurring the costs of moving equipment or using up consumable materials such as ammunition. Commanders can evaluate the behaviour of local populations in response to actions by deployed troops and select optimal courses of action. [1]

Simulation is used by scientific researchers to study the behaviour of biological systems, from cancer cell growth to insect population behaviour [2]. Using simulation and modeling allows researchers to control factors like the environment, observing the effect of changes in the environment on a system. Simulations can also be used to model natural disasters such as floods and forest fires. The behaviour of water or fire over time and different types of terrain is combined with information about the terrain in an area where the disaster is occurring or expected to occur. Execution of the simulation allows responders to predict the behaviour of the disaster and evaluate possible courses of action [2].

Simulation systems have been built for many years using many different types of technologies. CGF systems are an example of a category of simulation systems that are built using different technologies. OneSAF, a simulation system that can operate as a CGF or as Semi-Automated Forces, was written in Java using the Eclipse Integrated Development Environment (IDE) [3]. Another CGF system, VR Forces, has an Application Programming Interface (API) written in C++ [4]. While these systems are written in different languages, they have been developed with interfaces to allow them to communicate information with other systems. These interfaces use well-defined specifications such as Distributed Interactive Simulation (DIS) and High Level Architecture (HLA). These specifications describe simulated entities, their attributes, and how they can interact.

The ability for simulations to interact is important because more complicated scenarios can be built by having multiple scenarios work together [5]. One simulation system may model the behaviour of troops, while another may model the behaviour of civilians in the area. The use of standards such as HLA and DIS allows each component in a simulated environment to be modelled by a different system.

This allows the simulation to leverage the distinct strengths of each system. For example, simulated troops would be modelled behaving according to the orders issued by commanders. The behaviour of the citizens in the area may be modelled using an artificial intelligence system designed to model more generic human behaviour. As long as all of the systems in the synthetic environment comply with the communication standard being used they can share information and provide a more complex system under study or perform training with a higher level of realism for the students [5].

This need for standardized communication between different types of simulations has led to the development of a number of different standards. Standards such as DIS and HLA allow simulations to cooperate across multiple machines and computer domains [6]. Another category of technology that is used to communicate between multiple systems is Web Protocols. Web Protocols allow a client application on one machine to communicate with a Web Service that may be written in a different language, running on a machine located half-way around the world that uses a different operating system. Web protocols define the format of messages that are exchanged between systems and how the messages are exchanged. The Simple Object Access Protocol (SOAP) protocol uses Extensible Markup Language (XML) to define message content. Application Layer protocols such as Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP) are used to transmit messages [7]. The use of a known message format means that the client and server need not be concerned about how the messages are transferred in order communicate. The popularity of Web protocols for communications has led to them being used to communicate between simulation systems as well as real systems.

SOAP defines a format for exchanging messages by defining the header information attached to messages. However, it does not provide a definition for the message body, that is, the content of the messages being exchanged. SOAP in and of itself is not sufficient for defining a common message format that could be used between systems. The Common Battle Management Language (C-BML), on the other hand, is an XML-based specification that defines elements that can be used to create reports about simulated entities and their behaviours. The C-BML standard was developed to allow communications between Command and Control (C2) systems of different countries to exchange data. This standard also allows communications between simulation systems, as well as between Command and Control systems and simulation software. It has been developed based on the Base Object Model (BOM) that arose out of attempts to improve the use of HLA [6]. The Military Scenario Definition Language (MSDL) is another XML-based specification that uses many of the same elements as C-BML [8].

Instead of being used to define report message, MSDL is used to define the initial state information for entities in a simulation. It is used to create files that are loaded into simulations at the start of a simulation. The use of MSDL and C-BML allows the integration of real Command and Control software with different types of simulation systems to execute a large exercise in a single synthetic environment.

As mentioned, there are different types of simulation systems. CGF systems model the behaviour of troops and equipment in response to a commander's orders. Artificial Intelligence systems can model the behaviour of people in a large crowd. Another type of simulation system that can be used to model a wide range of scenarios is the Discrete Event System (DEVS) formalism. The DEVS formalism allows systems to be modeled as a series of discrete events occurring over time. It can be used to model a stimulus-response system such as a robot, and identify resource usage and bottlenecks in the modelled system. Cell-DEVS is an extension of the DEVS formalism that can also be used to model systems that can be defined by breaking the system down into multiple smaller systems. Cell-DEVS models take the concept of a single DEVS model and use them to build an array of connected models, where the state of a given model is calculated based on the state of the models adjacent to it according to the rules governing the behaviour of the model as a whole. Cell-DEVS can be used to model diverse systems, from the movement of ants to the electrical activity of heart tissue [9].

The range of capabilities of the DEVS modeling formalism represents a powerful tool for modeling and simulation. In order to make it more accessible for interaction with other systems the RESTful Interoperability Simulation Environment (RISE) server was developed [10]. The RISE server provides lightweight middleware for executing DEVS models. It provides an Application Programming Interface (API) that allows modellers to create DEVS models. This API provides an interface for interacting with an executing DEVS model. The invoking user can define the model, post the DEVS model, start its execution, and download the results. The interface provided by RISE is based on the Representational State Transfer (REST) web-service style [10]. One such system is the Web Service-Based Distributed Simulation of Discrete Event Models, as proposed in a paper by Colin Timmons [11]. It leverages this capability to interact as a part of a larger simulation.

As described, the RESTful interface defined by the RISE server enables clients to access models running on the server using web protocols. However, the RISE interface currently defines methods that allow a client to post a model and obtain the results of its execution. The server's interface does not allow a client to interact with a model running under RISE. Having the ability to interact with models executing

on the RISE server would allow the DEVS model to participate in a larger simulation over a longer period of time.

The goal of this thesis is to define an architecture which adds a simulation interoperability interface to DEVS models running on the RISE server. The intent of this architecture is to allow a DEVS model on the server to participate as part of a larger synthetic environment. The options and requirements for defining such architecture are studied. The architecture will be validated by creating a synthetic environment which uses two or more simulations executing concurrently to determine the outcome of a single scenario. The participating simulations will exchange state data with each other, where the state data represents the current or future state of one or more of the entities being simulated.

In order for a DEVS model to exchange data with another system, a method of communication must be defined. One of the goals of this thesis will be to examine the suitability of C-BML for sending reports of events in the simulated environment. In order for the simulations to exchange data they must agree on how they will communicate. Since RISE provides a RESTful interface, RESTful messaging will form the basis for communications. RESTful messages are well suited to XML message bodies. For this reason the use of C-BML will be investigated for sending messages to the DEVS model running under RISE. C-BML provides elements that can be used to create Reports about simulated entities.

Another goal of this thesis is to examine the use of the MSDL to initialize DEVS models running on the RISE server. The purpose of MSDL is to define the entities participating in a simulation and their initial state. MSDL and C-BML are used together in exercises for the purposes of initialization and communication respectively. This thesis will examine how data formatted according to the MSDL standard can be presented to and used by the DEVS model.

A related goal of this thesis is to examine the C-BML and MSDL standards for their suitability for use describing the interactions of civilian entities. These standards were created based on simulations performed by various military organizations. However, emergency scenarios such as fires and floods also involve civilian emergency service agencies, often working in concert with military organisations. The C-BML and MSDL specifications will be examined for their usability in modeling civilian entities and this thesis will present the results of this study.

The final goal of this thesis will be to validate the proposed architecture. To do this, a case study will be performed which models the behaviour of a civilian emergency service dispatch service. An application will be created to serve as a bridge between web-enabled constructive simulations and the RISE server.

The results of this case study will be examined to evaluate the performance of the proposed architecture.

1.2 Contribution

The main contribution of this thesis is the proposed architecture for adding simulation interoperability to the RISE server based on formatted messaging. The current interface provides some support for DEVS models on the RISE server to be created and used by web-enabled clients, but the clients must be designed and built specifically to work with RISE. Defining an interface that complies with interoperability standards would extend the possible scenarios in which the RISE server could be used as part of a larger synthetic environment.

As an example, the ability to model these types of systems could be used to enhance the accuracy of a planning exercise. A Cell-DEVS traffic model could be used to calculate the response times of entities that need to cross an urban center to respond to an emergency. A forest fire Cell-DEVS model could be used to determine the rate of advance of a fire, publishing the times at which the fire reaches key locations. This information could be supplied to the simulations that model the behaviour of other entities in the scenario, such as responders and commanders. Actions taken by other models in the simulation based on these inputs could result in changes to the overall scenario. Additional units may be tasked, or the event may be over before the responders can arrive. These changes may require re-calculation by the DEVS models based on the new information. This level of on-going collaboration between a DEVS model and other models requires that the DEVS model be interoperable with the other models. The contribution of this thesis is the architecture that makes this type of interoperability possible.

Another contribution of this thesis is an examination of how interoperability can be achieved for DEVS models running on a RISE server. The interface of the RISE server allows the creation and execution of DEVS models. An additional interface is required that serves as a bridge between the RISE server and other simulations using the interoperability techniques selected for this thesis: formatted messages and state information specification. A technique for creating formatted messages and supplying them to the DEVS model will be specified. As noted above, each time the DEVS model is executed it requires an up-to-date representation of the state of the entities in the simulation. A technique for identifying this state information and supplying it to the DEVS model will be specified.

One more contribution is the verification of the proposed architecture by implementing an application which conforms to the requirements for this architecture. The implemented system, the DEVS Bridge, is used to perform a case study. The results of this execution are examined, including the issues encountered while implementing the system and executing the case study. Among the results is a discussion of issues encountered when using the C-BML and MSDL standards for civilian simulations.

1.3 Thesis Organization

Chapter 2 presents the background material related to the topics in this thesis, including the technologies being examined. It describes work done in this field previously.

Chapter 3 presents the proposed Architecture for Distributed Simulation. It describes the proposed architecture in detail and the issues that will be addressed by the proposed architecture. It describes the use of C-BML and MSDL and issues with using them in this architecture.

Chapter 4 presents the implementation of the architecture and issues that were encountered.

Chapter 5 presents a Case Study that uses the proposed architecture. The Case Study is a simulation that models the behaviour of an emergency dispatch system.

Chapter 6 presents a discussion of the results of the Case Study.

Chapter 7 summarizes the work done and presents areas for future research.

2 Background

2.1 Simulation Interoperability

Computer simulation can be an effective way to analyze real-world situations. They are particularly useful in the case where holding a live exercise is not feasible due to the cost or limited availability of resources required, or in cases where safety prohibits holding an exercise. Numerous simulations have been developed over the years using a wide variety of technologies. Due to the time and effort invested in developing these simulations, the owners wish to continue to use them. These existing simulations may be used to model more complex scenarios by combining them with other simulations. Simulation interoperability becomes difficult as the newer simulations may be built using newer programming languages and networking protocols. In order to create a synthetic environment using multiple simulations, the systems need to be able to work together in an independent way. For this reason a number of standards have been developed for interoperability. The following sections discuss some of these standards.

2.1.1 DIS and HLA

The High Level Architecture (HLA) is a standard which provides a communication infrastructure to allow simulations to coordinate via formatted messages [1]. A simulation using HLA consists of individual simulations, known as federates, connected by a messaging infrastructure, known as the Run Time Infrastructure (RTI). The group of federates is referred to as a federation. Each federate models a number of entities. As these entities change their state, such as their location, or perform some action, the federate publishes a message onto the RTI describing the new status of the entity. Other federates receive this message and behave accordingly. All simulations connected to the RTI receive the messages published by individual federates. The published messages represent changes in the status of the objects in the simulation.

As an example, in a federation modeling the activity in a section of a city under military control, Federate A may model the actions of military organisations on patrol in the city, and Federate B may model the actions of insurgents who intend to ambush the patrol. Federate A publishes the current location of the patrol unit, and the direction and speed in which the patrol is moving. Federate B gets the information, and uses dead reckoning to calculate the movement of the patrol unit. Dead reckoning assumes that the patrol unit continues to move in the same direction and speed as was previously reported. Once the patrol unit reaches the location of the planned ambush, Federate B publishes

messages reporting the action of the insurgents, such as setting off an explosion. Federate A receives these messages and calculates the new behaviour of the patrol, such as damage from the explosion.

In order for the federates to interact, the objects in the individual federates must all be defined in a common Object Model, known as the Federation Object Model (FOM). The FOM defines the entities and their interactions. Through the use of FOMs over a number of years, common attributes were identified as being of use for all objects in a FOM, regardless of the type of environment being modelled in the federation. This led to the development of the Base Object Model (BOM) [6]. The BOM is discussed in more detail below.

The Distributed Interactive Simulation (DIS) standard is an older standard that served the same purpose as HLA. It defines a data delivery architecture with a strictly controlled message format. Instead of sending messages in a text-based format, as is defined in HLA, it defines Protocol Data Units (PDU) which specify message formats using binary notation for communicating between simulations [1]. Binary data is more efficient to process than XML, but more difficult to understand by human developers. With DIS the PDUs are broadcast, unlike HLA where updates are delivered only to simulations that subscribe to the RTI.

While DIS and HLA are both standards meant to allow Command and Control systems to communicate, they are not immediately compatible due both to the format in which data is exchanged, and the way the messages are exchanged. In order to allow DIS-compliant simulations and HLA-compliant simulations to work together in a single federation, a DIS Gateway may be used. The Gateway maps the broadcast DIS PDUs into messages that correspond to the Federated Object Model and publishes them on the RTI. The Gateway also performs mapping from HLA to DIS. To standardize the behaviour of Gateways the Real-time Platform Reference Federation Object Model (RPR FOM) was defined. This FOM organizes attributes and interactions of DIS models into an HLA hierarchy [12]. While it was defined for use in Gateways, it can also be used in federations that consist only of HLA-compliant federates.

2.1.2 Base Object Model

The Base Object Model (BOM) is a conceptual model of a simulated system. It facilitates interoperability, reuse, and composability by providing a common set of basic attributes [6]. These common attributes include features such as unique identifiers and other types of meta-data. The BOM also defines state machines and the events that trigger transitions within the state machine. It describes Patterns of Interplay, the actions that transpire between the entities in the model. [1]

Custom FOM definitions can be created by using the definitions in the BOM as base classes and adding attributes related to the domain. The BOM specification does not define the implementation of the Object Model, so it is independent of the language and technologies used to implement the models.

2.1.3 JC3IEDM

The Joint Command, Control and Communication Information Exchange Data Model (JC3IEDM) is a logical data model that defines concepts that are common in a Command, Control and Communications (C3) environment. The model was developed by the Multinational Interoperability Program (MIP), a consortium of twenty-nine NATO and Non-NATO nations. The goal of the MIP is to promote international interoperability of Command and Control Information Systems at all levels of command [13]. The purpose of defining a logical data model was to allow international Command and Control (C2) systems to communicate in a way that is independent of the language spoken by the users of the C2 systems. Each member nation implements objects representing the data entities in their own C2 system. These C2 systems exchange data in Protocol Data Units and other binary formats. Communications between C2 systems is done by replicating data between databases using PDUs or other similar message formats. When the connections between the databases are set up, the rules are set up for how data will be exchanged between them.

The JC3IEDM was initially designed to capture data required for Command and Control operations. Specialized functional areas such as fire support operations served as sources of requirements for the original development of the data [14]. The intent was to allow commanders to send instructions to units, and allow the units to report back observed entities and their status. The data model is fully normalized [14]. This means that identifying one item, such as a unit, requires the creation of multiple records, such as location, item-type, and item-status, which are linked together via a single identifier. The entities include elements for units (armies, platoons, non-government agencies, individual persons, etc.), equipment (trucks, tanks, mortars, etc.), facilities (buildings), terrain (point locations or polygons), and weather conditions. Most fields in the records are defined using Category Codes. Each Category Code consists of values that attempt to capture all possible values for that Category Code. For example, the Affiliation Ethnic Group Code has over 500 values, including Not Otherwise Specified (NOS) and Not Known (NKN).

In addition to entities, the JC3IEDM specification defines activities, or Actions. The specification divides category codes for actions into two types: Action-Tasks, which are instructions for units being tasked by a commander, and Action-Events, which are activities which are observed being performed by entities

outside the control of the commander, such as civilians, hostile forces, and non-government organizations.

The JC3IEDM defines the logical model. Member nations that use it implement their own physical databases. Objects in these databases persist throughout the duration of the exercise or deployment. The current state of the object is labelled. This may have an impact on the performance of the database over time.

2.1.4 Coalition Battle Management Language (C-BML)

In contrast to JC3IEDM, where data is exchanged by database, the Coalition Battle Management Language (C-BML) standard is used to exchange battle management doctrine in the Command and Control (C2) environment using formatted messages at the application layer [1]. The standard seeks to express the commander's intent in an unambiguous fashion using XML messages. It is defined based on the JC3IEDM [15] and uses many of its Category Codes directly.

The C-BML standard is not a specific schema. Instead it is a set of building blocks that can be used to define message formats to be used during an exercise. The messages can be exchanged between live forces, simulated forces and robotic forces all within the same exercise, thus allowing simulations to interact with live operators [1]. The entities defined in the C-BML standard capture the five W's of information: Who, What, When, Where and Why [15]. The standard uses many of the entities defined in the JC3IEDM model, including Units, Actions and many of the Category Codes from the JC3IEDM. However, it uses a subset of the entities, meaning that the messages created using C-BML may be smaller than those required by the JC3IEDM [15].

The entities described in C-BML messages are described below. Their use varies depending on the message type.

- Where – Describes Location, Route, Geographic features, etc. The location may indicate where a unit should move to, or where an Action took place.
- When – Describes either relative time, e.g. unit should move x amount of time after an event, or absolute time, e.g. an event occurred at time x.
- Who – Describes actors performing an activity.
    - The Who entity may represent:

        - An Organisation – A group of persons, such as a unit, a company, an Army, or a Non-Government Organisation.
        - Person - An individual.
    - Depending on the type of message, Who may be:
        - Reports: Reporter, Addressee, Reported on
        - Orders: Tasker, Taskee, Affected by task
- What – Depending on the type of message, What may be:
    - Reports: Action that takes place
    - Orders: What the tasked unit needs to do
    - There are two main types of Activity:
        - Action Event – something that happened outside the control of the planning process, e.g. explosion. Specified as Action Event Category Codes.
        - Action Task – something an entity under the control of the planning process is being tasked to perform. Specified as Action Task Activity Codes.
    - Category Codes for these are defined in JC3IEDM. They are included as part of the C-BML Phase 1 definition.
- Why – Rationale for performing the task.

Objects that are produced during a Command and Control exercise are never updated or deleted. Instead, updates are created for the objects. For this reason they can be modeled as web resources. This makes C-BML suitable for implementation using Representational State Transfer (RESTful) web services [16]. REST is discussed below.

The standard is under development by the Simulation Interoperability Standards Organisation (SISO) in three phases. Phase 1 is complete, and specifies the data model. Phase 2 is aimed at formalizing the grammar, and Phase 3 is meant to formalize the ontology [1].

2.1.5 Military Scenario Definition Language (MSDL)

The Military Scenario Definition Language (MSDL) is a standard for specifying the start state of entities in a scenario. It can be used to describe organisational hierarchies, initial deployments, terrain and weather, and plan objectives. It is closely related to C-BML [1]. It reuses the Base Object Model (BOM) standard and JC3IEDM category codes. It has been used to initialize systems in exercises where C-BML is used for communications, such as the NMSG-085 Land Operation demonstration [17].

The MSDL format is used to create files that may describe the force structure, or ORder of BATtle (ORBAT), and how organizations are specified, either as individual entities, or as aggregates, e.g. units. MSDL files may include units, equipment or installations/locations that are key to the exercise. It may be used to specify plan objectives, the environment, including scenario time, terrain and weather. The specification includes syntax that may be used to describe activities of entities, including Military Operations Other Than War (MOOTW), which include Sniping, Riots, and forcible recruiting of locals[1][18]. Unique Ids for described entities are then used to refer to them in the C2 messages, which may be defined using the C-BML standard.

2.2 Simulation Exercises

2.2.1 C-BML Server

The Command, Control, Communications, Computing and Intelligence (C4I) Center of Excellence at George Mason University (GMU) has developed components for use with C-BML. They have created a web service that serves as a repository for C-BML messages, including orders, reports and requests. The messages are packaged as XML documents and stored in a relational database [19]. The default database is based on the JC3IEDM specification, but the database may be altered using scripts. The Service Manager uses a subscriber, push/pull mechanism for managing data. Orders and Reports are pushed to the database [20]. Entities or federates involved in the exercise, subscribe to notification of data pushes. The entities can pull data from the database via the Service Manager when notified of data of interest.

GMU has also developed the Scripted BML Server (SBML), which is a form of middleware that is intended to allow rapid development of web services as BML evolves [19]. SBML allows the database to be tailored using scripts rather than reprogramming. This makes the database more robust and facilitates faster development when there are changes to the protocol. Since C-BML provides building blocks for defining a specification, a specification that is used for a particular exercise may well change at any time leading up to the exercise as the objectives of the exercise evolve. Using a script to define mapping into the database facilitates adjusting to changes in the schema, and makes it easier to debug the resulting database and Service Manager [20][21][22]. It also allows rapid development of the Web Service as the model evolves, since only the scripts need to be changed as the C-BML schema evolves [23].

The SBML provides a RESTful interface as well as a SOAP interface. The RESTful interface was found to provide a 15% improvement in performance versus the SOAP-based server during trials [19]. The RESTful server allows clients written in programming languages other than Java to create subscriptions to notifications of new data as long as they have access to an HTTP library. The SBML server was used in the NATO MSG-048 experiment [19] and is being used in NATO MSG-085, which is scheduled to produce its final report early in 2014 [17]. Preliminary findings are reported in [17].

The C-BML Server configuration is shown in Figure 24 below. The C2 applications communicate with their local BML Server. The BML Server exports data to the databases. The database schema may be the full JC3IEDM, or a custom schema adapted to the needs of the exercise. New data is shared between the databases in the exercise, based on rules of data replication. When new data is received by a database, it notifies its subscribers, who may then request full data.

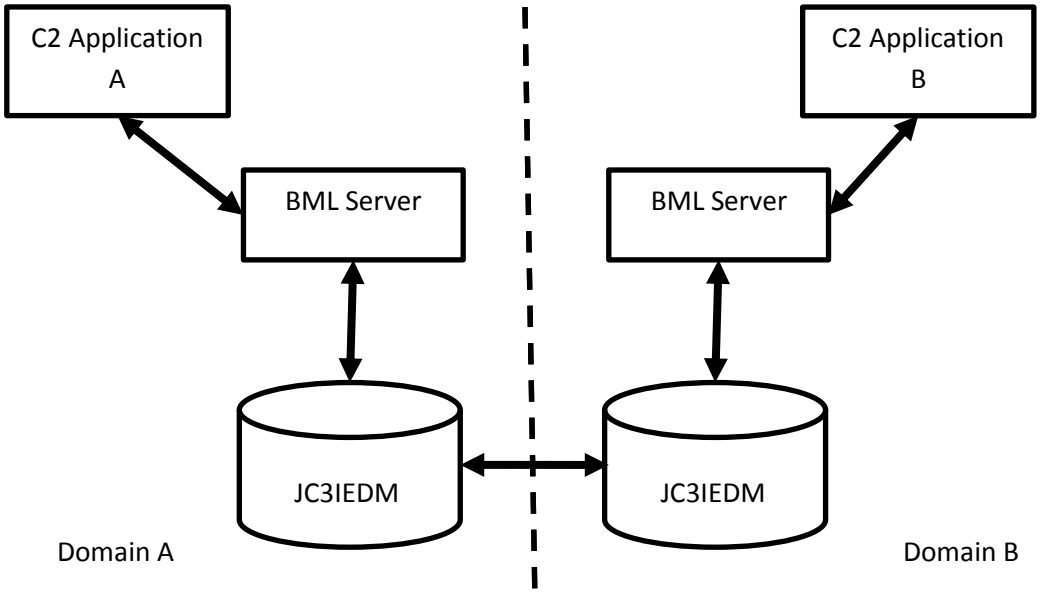

Figure 1 Typical C-BML Server Deployment

As an example, a commander in Domain A creates orders for Unit B, which uses computers in Domain B. The commander enters the order using C2 Application A. The order is pushed to the BML Server in Domain A, which stores the order in the database. According to the replication rules between the two databases, the order is copied to the database in Domain B. The receiving database raises a notification to the BML Server. C2 Application B has subscribed to notifications of incoming orders. It receives the

notification of new data and pulls the new order from the database. Members of Unit B retrieve the orders from C2 Application B.

2.2.2 Phase 1 Trial Use Results

The draft C-BML specification schema was provided for a community trial use. It was found that processing received C-BML expressions is relatively complex due to the fact that XML elements are nested and cross-referenced by other elements, and there are circular references. The resulting messages may require multiple processing phases before their contents are fully understood [16].

The Norwegian participants in the exercise implemented a proof-of-concept C-BML report framework. They provided support for parsing, validating and persisting C-BML expressions. However, as of 2011 there were no plans to implement a full Information Exchange Model with the framework developed for the exercise [17].

2.2.3 Initial Demonstration of use of multiple specifications

A Use Case for an initial demonstration of using multiple specifications is given in [1]. The mission involved autonomous robotic forces that require detailed instructions. The robots were part of an Anti-Terrorism/Force Protection (AT/FP) of a harbour. The vehicles included Unmanned Surface Vehicles, (USV), Unmanned Arial Vehicles (UAV), Unmanned undersea vehicles (UUV) and Unmanned Ground Vehicles (UGV). They patrolled the area being monitored and alerted response teams. The following standards were used:

- BOM – define entities and interactions
- C-BML – orders given to platforms
- DIS-XML – runtime state updates, visualization of scenario, logging of state changes, entity messaging

The efforts that went into the design of the use case became input into the evolution of the Phase 1 Specification of C-BML [16].

2.2.4 NMSG-085

Another exercise forming part of the Phase 1 Trial Use of C-BML included the NMSG-085 exercise. It was based on demonstrations carried out between Denmark, France, Germany, the Netherlands and Spain [17]. The demonstration included both simulated and operational C2 systems working together. The

simulated forces were run using VR Forces by MaK Systems. VR-Forces is a Computer Generated Forces (CGF) application.

The exercise used MSDL to initialize the scenario. All simulated C2 systems sent an MSDL file containing the Orders of Battle (ORBATs) of the units they managed to a central system. The central system merged all of the files into one global ORBAT. This ORBAT was then distributed to all of the participating C2 systems. This was the first use of an effective initialization process based on the MSDL standard. As part of the lead-in to the experiment MSDL was extended. Extensions included defining unit types with JC3IEDM dictionaries, and using the Allied Procedural Publication 6A (APP-6A) standard for definition of unit symbols.

During the exercise C-BML was used for the exchange of logistics. All C2 systems created and retrieved their expected reports in their native formats. Reports were created for unit positions and status, and munitions explosions. During the exercise officers built orders for subordinates. Among the messages that were created were requests for artillery calls for fire and acknowledgement of the calls. The presence of systems from multiple nations using their own C2 systems highlighted the suitability of C-BML for this type of data exchange. Recce units were disaggregated and transferred to VR-Forces as well. This demonstrated transfer of ownership of simulated entities, which is a key feature of DIS/HLA simulations.

During the exercise, the C-BML server built by GMU was used for exchanging messages with some modifications. Instead of using the full JC3IEDM database storage simple XML file storage was used, which reduced the maintenance costs of the SBML scripts.

The conclusion of the report authors was that successful execution of the exercise demonstrated that C-BML is well suited to C2 to C2 data exchange [17].

2.2.5 Civilian Simulations

Militaries are not the only agencies that can benefit from the simulation of large scale operations. Civilian emergency response agencies need to be able to deploy resources in response to natural disasters or other emergencies. Often these agencies interact with military agencies [25]. Simulation has been used to investigate asset utilisation and planning when the military has been deployed as backup in the case where fire fighters unions went on strike [26]. This type of simulation has become more important since the events of September 11, 2001 [27]. Simulation and other models are being used for Emergency Management as documented in [28] and to simulate natural disasters [29]. DEVS simulation

is also being used for aspects of health-care modelling including the management of EMS services [30] and triage simulation [31].

2.3 Discrete Event Modeling

2.3.1 DEVS

Discrete Event System Simulation (DEVS) is a technique for modeling the behaviour of systems as a series of behaviours in response to stimuli. The system is broken down into a set of one or more atomic models. Each atomic model is defined in terms of the states, inputs and outputs of the model. Responses may be immediate, or the model may pause for a period of time before an internal transition generates the reaction of the system. Multiple atomic models can be coupled to form larger models.

The DEVS formalism is of an atomic model is expressed as

$$M = \langle X, Y, S, \delta_{inp}\ \delta_{exp}\ \lambda, ta \rangle$$

where

$X = \{(p,v) \mid p \in IPorts, v \in Xp\}$ is the set of input events. IPorts represents the set of input ports and Xp represents the set of values for the input ports;

$Y = \{(p,v) \mid p \in OPorts, v \in Yp\}$ is the set of output events, where OPorts represents the set of output ports and Yp represents the set of values for the output ports;

S is the set of sequential states, where the state may be the value of data held by the model;

$\delta_{ext} : Q \times X \rightarrow S$ is the *external* state transition function in response to an external stimulus, with $Q = \{(s,e) \mid s \in S, e \in [0,ta(s)\}$ and *e* is the elapsed time since the last state transition;

$\delta_{int}: S \rightarrow S$ is the *internal* state transition function;

$\lambda: S \rightarrow Y$ is the *output* function; and

$ta: S \rightarrow R_0 + \cup \infty$ is the *time advance* function. [32]

2.3.2 CD++ Tool

The CD++ tool is a plugin to the Eclipse development platform. It supports the definition and execution of DEVS models using C++. Each atomic model is defined as a subclass of the Atomic class. Input and

Output Ports are defined as member variables of the atomic models. The methods in the DEVS formal definition, as well as an initialize function, are inherited from the Atomic class and must be overridden in the subclasses. The methods that need to be overridden are:

- initFunction – used to define initial values for the model attributes
- externalFunction – implements the external transition function. It is invoked when inputs are received on an input port from another model. It may send output immediately, or it may schedule an internal transition after a delay, and provide output at that time.
- outputFunction – implements the logic to generate outputs. Invoked when internal transitions occur after the timeout value, before the internalFunction is invoked.
- internalFunction – implements the internal transition function. It is invoked when an internal transition is scheduled to occur.

The overall DEVS model is defined in an MA file. The file defines a top model, which is a coupled model made up of instances of the atomic models and other coupled models. The MA file defines:

- instances of atomic models
- input and output ports of the coupled model
- connections between the ports on the atomic models, from output ports to input ports
- coupled models; and
- values for variables that are used by the atomic models.

A model may also define an events (ev) file. The events file contains values that are sent to input ports defined on the top model, and the times at which the events are to be input. Execution time is simulated; the time at which an event is scheduled to execute is used to determine the order in which events are to be processed. The CD++ framework instantiates the atomic models and delivers the events to the models via the ports. Two files are output from the model execution. A log file traces all messages between all of the ports in the model. The out file displays outputs from the ports on the top model, with the time at which the output was generated and the name of the port [2].

2.3.3 Uses of DEVS

The modular nature of DEVS models and its closure under coupling has led to its applicability in a number of fields. It has been used to create a collaborative environment for developing HLA-compliant federates [32]. DEVS has also been used to build testing and evaluation systems that can be accessed via

web technologies [33]. DEVS simulations have been used to evaluate the behaviour of modelled systems such as cellular networks [34]. Cell-DEVS has been used to model biological systems, such as the growth of cancer cells, and physical systems such as heat diffusion [2].

2.3.4 Other types of DEVS models

In addition to the basic CD++ tool, there are other DEVS modelling tools that provide additional capabilities:

- Embedded DEVS, or Real-Time DEVS, run in systems where there are time constraints.
- Parallel DEVS runs models on multiple processors, coordinating between them.
- Cell-DEVS models systems as a series of models in parallel, modelling systems that can be represented as executable cell spaces [2].

2.3.5 RISE

The RESTful Interoperability Simulation Environment (RISE) provides a framework for executing DEVS simulations. It provides a RESTful interface that allows the creation, execution and examination of simulations. Currently the RISE server supports Parallel DEVS and Cellular DEVS.

RISE is a server with namespaces, arranged in a hierarchy. The root of the server is accessed via the Uniform Resource Identifier (URI) as <machine-URI>/cdpp/sim/workspaces. Under this level workspaces are created for individual users. Under the user workspace further workspaces may be created for each type of DEVS model. At the time of writing two types of models were available: dcdpp for Parallel models, and Lopez for cellular CD++. Models are executed by POSTing them to the RISE system under an appropriate namespace. The model must be completely defined so that it can execute autonomously, with no resources from outside the server [35].

The RISE server currently supports two types of DEVS models: Parallel CD++ and Distributed CD++. Models are created on the server by sending a PUT message with an XML file defining the model. The XML file defines the names and types of all files in the model. File types may be one of several types:

- C++ header files (hdr) defining atomic model classes
- C++ source files (src) defining the method bodies of the atomic model classes
- one ma file (ma) defining the coupled model(s)

- event files (ev) defining events that are to be input to the top model
- supplemental files (sup) such as readme text files.

For Parallel DEVS models the XML file identifies the servers used in the simulation. The file specifies which server the model instances run on, e.g. instance barber1 of Barber runs on Server1, and instance barber2 runs on Server2. The instance names must map to the names of the instances defined in the ma file.

Non-parallel models that run under CD++ can be run under dcdpp by specifying only a single processor. The source code does require a minor change: the argument type of the outputFunction must be changed from an InternalMessage to a CollectMessage prior to being posted to the server.

Once the model has been defined on the server, the source files must be compressed into a zipped archive file. This file is then posted to the server. The model is executed by accessing the "/simulation" resource under the model name. If the model executes successfully the out and log files can be accessed under the "/results" resource. If the execution fails or runs into an error compiling the error files can be accessed under "/debug". The results and the logs are returned from the server in a zipped archive file.

The RISE framework allows users to execute simulations across a web interface, making them accessible from any location. This principle has led to the development of support for running simulations in the cloud, including a proposed software architecture for use by emergency crews. This architecture leverages RISE and Taverna Workflow software [36].

2.3.6 DEVS Model Interoperability using HLA

Work involving HLA and DEVS has been done on a number of fronts. DEVS models have been mapped to HLA to automate the programming work for constructing HLA compliant simulations [37]. A DEVS/HLA distributed simulation environment was developed for the Defense Advanced Research Projects Agency (DARPA) to study quantization as a basic approach to predictive filtering. It will support study of more advanced predictive contract mechanisms. However, there are some issues with time management. Discrete Event Simulation is based on logical time, but HLA is oriented towards real-time simulations. A test case was developed using the PARSEC DESL language to study these issues [38].

DEVS and HLA have been used in combination for other purposes. The HLA RTI has been used as the DEVS Root Coordinator to communicate between models in a distributed DEVS model. This technique was used to develop a tool for designing and developing services [39]. It has been used to develop a

distributed discrete simulation environment based on Generalized Discrete Event System Specification (G-DEVS) combined with HLA [40]. Other work done with G-DEVS and HLA includes developing a flattened simulation structure to define new workflow processes. The workflow reference model confirms to the HLA standard [41]. There has also been work done to evaluate look-ahead methods using GDEVS federates in an HLA federation [42].

2.4 Representational state transfer (REST)

2.4.1 Overview

Representational state transfer (REST) is a style of software architecture for distributed systems. Early web services developed custom interfaces, with separate methods for every operation performed by the service. As a web service expanded and offered more operations, more interface methods were required, complicating both the design of the service API and the knowledge required by the clients of the service. The protocols used for web service interaction had to accommodate the possible complexity of new and existing services [43].

The intent of the REST interface was to provide a simplified protocol. REST emphasizes scalability, generic interfaces, and independent deployment of components [43]. Instead of defining a protocol that can accommodate a growing set of message types, the protocol uses only four of the HTTP message types: PUT, POST, GET and DELETE. All operations performed by the web service are modelled as changes to resources defined on the service. The resources are identified through a Universal Resource Indicator (URI) which identifies both the web service and the resource on the service. Messages to the Web Service are created as HTTP messages. The type of the message is one of the four HTTP message types identified above. The message is sent to the URI for the resource. Additional information may be included in the HTTP message body.

2.4.2 Uses of REST

The use of REST as a method of connecting clients and services via a technology-agnostic interface makes it applicable to a number of fields. The simple interface makes it a good fit for efforts to bring together data from globally disparate sources, where users may have different levels of resources and technical support available. These reasons led to its use in the NASA Sensor Web, which consolidates web sensor data for the purposes of disaster management [44]. Similar differences in sensor types led to its being used in other web sensor networks [45].

Another example of using REST in the collection of data from multiple sources is in the construction of a map mashup, where a RESTful service provides railway data that is combined with map data to provide geographic context to users interested in railway information [46].

The simplicity of the REST interface compared to the traditional SOAP interface has led to efforts to access current services via a simpler interface, for the purpose of developing a semantic web [47]. RESTful techniques are also being used as part of efforts to develop mashups of existing services as described in [48] [49] and [50].

3 Architecture for Distributed Simulation using RISE and C-BML

3.1 Overview

3.1.1 DEVS Bridge description

The intent of this thesis is to develop and validate an architecture that allows DEVS models to participate with other simulation systems to create a larger synthetic environment. In such an environment the DEVS models would manage the behaviour of simulated entities, or provide information that affects the behaviour of entities managed by other simulations. In order for this to happen, two basic capabilities are required:

- an environment where the DEVS model can execute; and
- the ability to communicate with other simulations in the synthetic environment.

The first capability is provided by the RISE server, which hosts workspaces where Parallel DEVS and Cell DEVS models can execute. What is required is the ability to interact with other simulations: receive output from these simulations as input to the DEVS model, process the input, possibly using local state information, and then share the output from the DEVS model with the other simulation(s).

This thesis proposes that one way to achieve the second capability is to define a new service, the DEVS Bridge. The DEVS Bridge acts as an adapter, allowing the DEVS model on the RISE server to connect to the larger environment and react to events that occur in that environment. The responsibilities of the DEVS Bridge are:

1. subscribe to messages being published by the other simulations in the environment,
2. translate these messages into a format native to the DEVS model,
3. trigger the DEVS model to process the input,
4. collect the output from the DEVS model,
5. translate the output from the DEVS model to the format being used in the larger environment; and
6. maintain any state data required by the DEVS model between executions, e.g. current location of simulated entities

The use of the DEVS Bridge allows the DEVS models to remain agnostic as to the format being used for simulation interoperability. For example, the DEVS Bridge could implement a federate using HLA to

communicate with other federates as described in section 2.3.6, or could subscribe to an operational database using the JC3IEDM schema as described in section 2.2.1, receiving notifications of new information and publishing the results of the DEVS model execution to the database.

For the purposes of this thesis it is assumed that the synthetic environment is one in which communication is performed by publishing C-BML messages. The DEVS Bridge built for this thesis is designed to receive information as XML-formatted messages which are to be parsed, translated, and presented to the DEVS model as input.

The proposed architecture is shown in the figure below. For the sake of simplicity the workspaces on the RISE server are not shown.

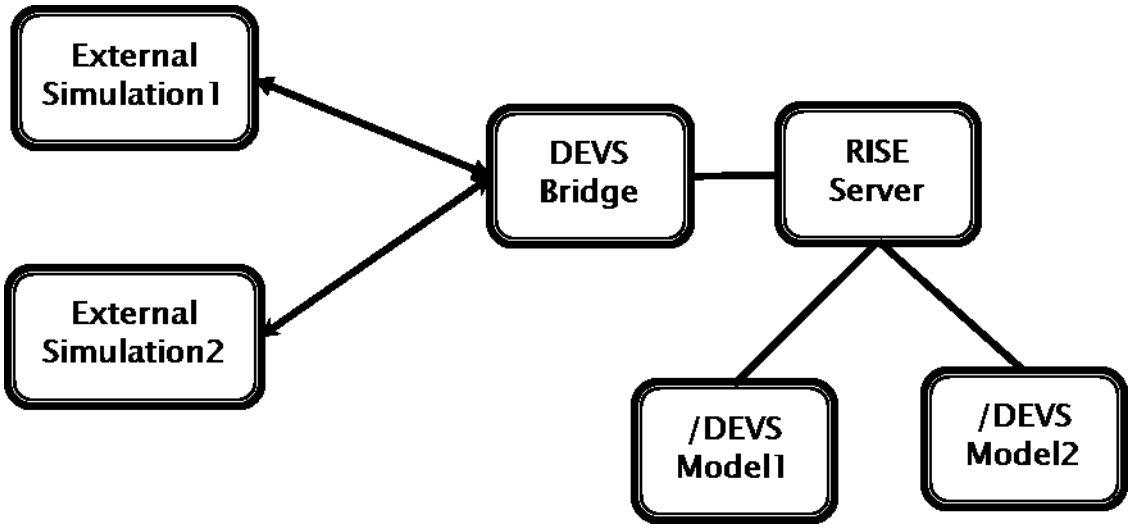

Figure 2 Architecture Overview

In addition to its responsibilities during the execution of the scenario, the DEVS Bridge also has responsibilities related to the setup of the scenario prior to execution. It must participate in any system-wide initialization, including loading data related to the simulated environment such as locations of interest in the scenario or weather conditions. It must also load initial status information related to the simulated entities such as their locations, equipment holdings and reporting structure.

In addition to these responsibilities the DEVS Bridge also has some responsibilities that are unique to initializing the DEVS environment, including setting the model up on the RISE server prior to execution. These responsibilities are discussed in more detail in the following sections.

3.1.2 RISE server description

As mentioned, the RISE server provides the environment where the DEVS models execute. Its responsibilities include:

- hosting the DEVS models and their related data files,
- executing the DEVS models; and
- making the results of the model available once execution is complete.

The RISE server provides a RESTful interface that is used by the DEVS Bridge to access the DEVS models.

3.1.3 DEVS Model description

The DEVS models hosted on the RISE server can simulate a wide range of scenarios. Using the receipt of data from the external simulations as an event to trigger execution, the DEVS model can calculate the response of an automated robot to commander's instructions, or calculate the movement of fire across an area in response to changes in the environment such as the presence of fire fighters or water bombers being simulated by another system.

Communications between the components in the architecture during a possible scenario are shown in the sequence diagram below. The diagram assumes that the scenario is being controlled by one of the external simulations.

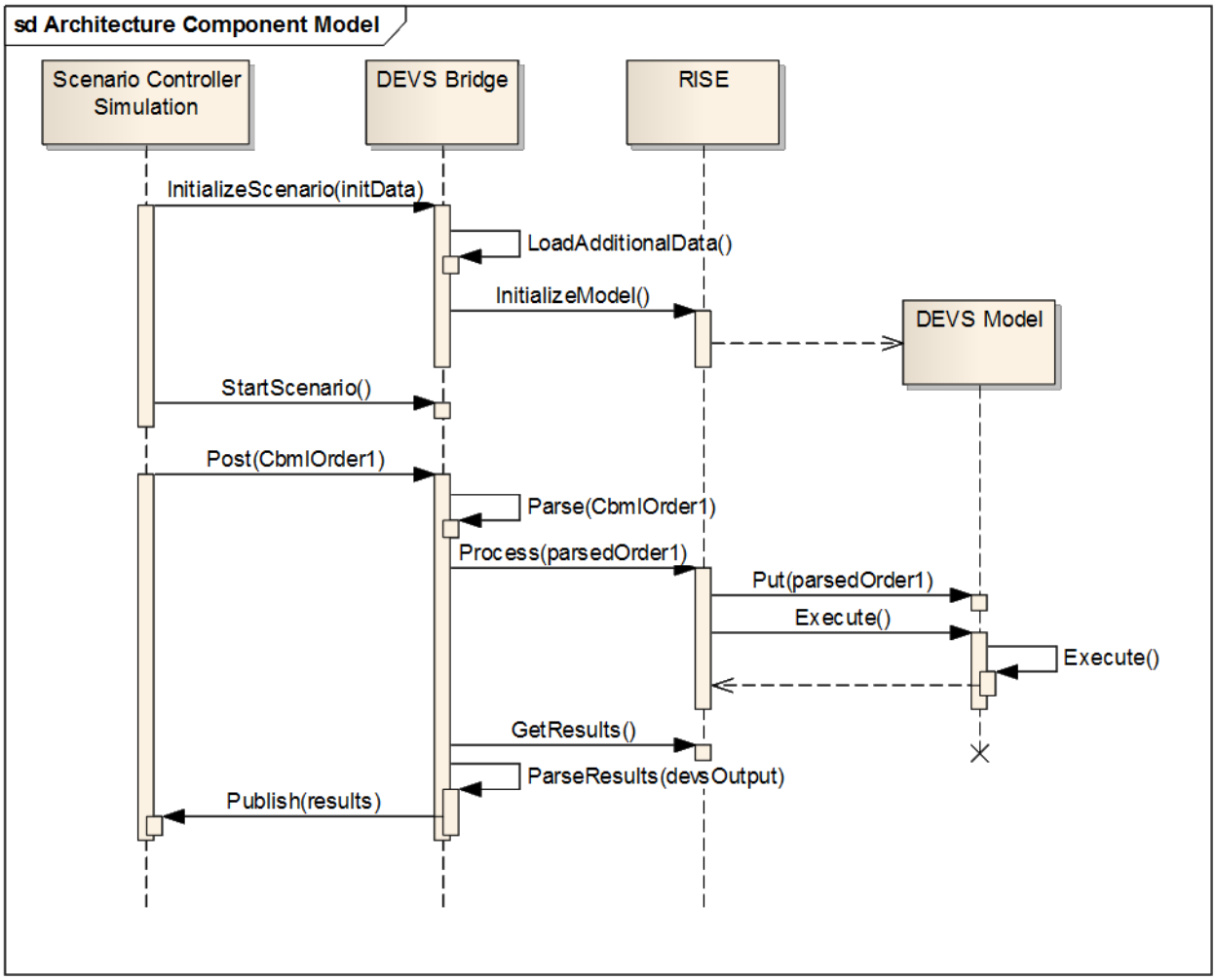

Figure 3 Messaging between components at the architectural level

The figure shows messaging between the major components in the architecture in a typical scenario.

- The external simulation which is functioning as the Simulation Controller sends out an Initialize Scenario message, which includes any initialization data required by all of the simulations participating in the scenario.
- The DEVS Bridge receives this data, and loads any additional data it requires. It also initializes the DEVS Model on the RISE server.
- The external simulation sends a message indicating the start of the scenario.
- As the scenario progresses, the Simulation Controller sends a C-BML Order intended for the entities managed by the DEVS Model. The DEVS Bridge detects/receives notification of the order.

- The DEVS Bridge parses the C-BML Order into the native format of the DEVS Model.
- The DEVS Bridge supplies the Order to the DEVS Model on the RISE server.
- The DEVS Bridge triggers the execution of the DEVS Model.
- The DEVS Model calculates and outputs the response of its managed entities to the received Order.
- The DEVS Bridge retrieves the results of the DEVS Model execution.
- The DEVS Bridge maps the DEVS Model output into a C-BML report.
- The DEVS Bridge publishes the DEVS Model output in C-BML format.

The remaining sections in this chapter discuss the DEVS Bridge design and capabilities in more detail.

3.2 DEVS Bridge Capability Overview

In addition to the capabilities required for coordinating messages between the external simulation and the DEVS model, the DEVS Bridge has a number of other responsibilities that are required to implement the architecture. These responsibilities are required in order to ensure the smooth operation of the DEVS Model during the simulation.

3.2.1 Scripting Function

The purpose of this thesis is to demonstrate an architecture for making a DEVS model on RISE interoperable with other simulations. In order to satisfy this requirement, the DEVS Bridge requires an interface that can receive messages from an external system, and another interface that can send messages back to the external system. The capability to exchange text-based messages via web protocols has been well-documented using a number of different message formats, such as C-BML and JSON, and using different protocols, including HTTP, SOAP, and others. Thus, proving these capabilities is not required to demonstrate the viability of the architecture proposed in this thesis.

While it is not necessary to actually have an external simulation as part of the demonstration of the architecture, it is necessary to have some stimulus for the DEVS Bridge to trigger its interactions with the model on the RISE server. For this reason, it was decided to use a script-based interface. The script contains the messages that would be received from the external system. Each step in the execution of the simulation is triggered by the processing of the next message in the script.

The use of a scripting language to trigger the behaviour of a system is another technique that is well-proven [51][52]. For that reason, rather than implement a separate scripting tool, the scripting

capability was built into the DEVS Bridge directly. The DEVS Bridge has capabilities to create a script, save it to file and load it from file as well as execute the script one step at a time. Having the ability to store and export the set of messages received from the external simulation is a reasonable functional requirement on the DEVS Bridge for other reasons as well, as it provides a capability to record the execution of the simulation. This capability allows after-action review of simulations, which is a key feature of many simulation systems [5].

3.2.2 State management and Message/Resource State Function

The execution of a parallel or Cell-DEVS model on the RISE server runs in simulated time, running from a given start state to an end state which provides information to the larger simulation about the entities modelled by the DEVS model. In response to this information, entities in the larger simulation may take specific actions, resulting in the need to execute the DEVS model again to determine the new behaviour of its modelled entities. Between these two executions of the DEVS model, the state information describing the entities modelled by the DEVS Model needs to be preserved so that it can be provided as input to the next execution of the DEVS Model. The output from run n needs to be provided as input to run n+1.

For this reason the DEVS Bridge needs to be able to manage the state information for the entities modelled by the DEVS model. This requires the DEVS Bridge to have its own copy of the data model used by the DEVS model. When the scenario starts, the DEVS Bridge needs to load the model and the initial state information, e.g. number of units, locations of units and their initial equipment holdings, locations of interest that may be referenced during the scenario, environmental factors of interest, e.g. weather conditions, and other factors. As messages are received from the external simulation, the DEVS Bridge must map the reports and requests to entities in its data model. This is partly to determine whether the received message applies to its internal model, in the case where reports are broadcast to all simulations in the synthetic environment rather than targeted only to simulations that have subscribed to notification. It is also done so that the DEVS Bridge can determine which of the many entities it manages need to be updated or referenced in the data generated for input to the DEVS Model.

When the DEVS Bridge determines that it has received a message pertaining to the entities managed by the DEVS Model, it needs to be able to generate all required input to the DEVS Model. This includes formatting state information messages for the DEVS Model so that all of the atomic models may be initialized properly prior to processing the received message from the external simulation. This state

information is created based on the current state of the data model held by the DEVS Bridge. After the DEVS Model has executed, the DEVS Bridge must update its internal representation of the data model state, and then use this information to generate the response to the original message from the external simulation.

3.2.3 Model creation function and why it is required

The RISE server provides an environment for creating a workspace for a DEVS Model and executing the model. However, this model must reside on the server before it can be executed. The location of the model must be known to the DEVS Bridge. Therefore, the DEVS Bridge has the capability to create the workspace on the RISE server, post the model to the server, and get the results of the execution. For the sake of convenience, all general maintenance operations required to manage a model on the RISE server are built into the DEVS Bridge, including the ability to delete the model once the simulation is complete. General environmental setup and clean-up abilities are requirements of many simulation systems [5].

3.3 DEVS Bridge Design

3.3.1 Scripting Function Design

As described in section 3.2, the DEVS Bridge was designed with a number of capabilities which are required to execute the scenario. These include the scripting function, model creation function and state management function.

Before the scenario can start, the DEVS Bridge must have a script containing the messages to be sent to the DEVS model. If the script has been created previously, it may be loaded from file. If not, the script may be created via the Message Set editor.

The Message Set editor displays the list of messages to be executed in the scenario, in the order in which they are executed. The Message Properties editor allows the user to create new messages or edit existing ones.

A Message Set object contains the set of Message objects for the scenario. The entire Message Set can be converted to XML in order to be saved to file. The XML file can later be read-in from file and

converted to objects for use by the DEVS Bridge. Saving the Message Set to a file allows the same scenario to be executed without having to recreate it each time.

During execution of the scenario the DEVS Bridge steps through the Message Set in order. The current Message is serialized in the format used by the DEVS model and used as input to the model for the current execution of the model. Once the model has returned the results from processing the message, the DEVS Bridge advances the current message reference to the next message in the list.

3.3.2 State Management and Message/Resource State Function

In addition to a set of messages, the DEVS Bridge requires a set of resources in order to execute a scenario, as described in section 3.2.2. The DEVS Bridge was designed with the capability to manage the resource set.

Before the scenario can start, the DEVS Bridge must have loaded a set of resources. Resources consist of a list of locations and units that are relevant to the scenario being executed. If a resource set has been created previously, it may be loaded from file. If not, the resource set may be created.

The resource set must be created before the message set for a scenario is created, as the message set uses the unique ids of the locations in the resource set in the messages. A resource set can be re-used by multiple scenarios, but a scenario can only be used with a particular resource set. However, the resource set and message set need not be loaded into the DEVS Bridge in any particular order.

The Resource Set editor displays the list of locations and the list of units. The Location Editor and Unit Editor allow the creation of new items or the editing of existing items. Because Units have Locations (see section 3.7 Data Model) the Unit Editor displays a list of all currently-defined Locations. The selected Location becomes the Location of that Unit at the beginning of the scenario. As the scenario progresses the Location of the Unit is updated based on the output from the DEVS model.

The entire Resource Set can be converted to XML and saved to file for use later. The XML file is read in and converted to objects. The Resource Set can also be converted to the custom text format used to communicate with the DEVS Model. During execution of the scenario, when the DEVS Bridge processes a message from the script, it converts the entire Resource Set to the custom text format used by the DEVS model. The DEVS Bridge passes the formatted version of the Resource Set to the DEVS model, which uses the Resource Set to initialize the atomic models.

Once the DEVS model has completed processing the DEVS Bridge parses the output and uses it to update its local copy of the Resource Set. This involves updating the current Task and current Location of the units. See section 3.7 for more information on the Data Model used in this experiment.

3.3.3 Model Creation Function

In addition to managing the flow of the scenario script and updating the status of the Resource Set, the DEVS Bridge also provides a capability for posting the model to the RISE server. In order to post a model to the RISE server, two files are required:

- an XML file describing the model including the list of source code files, the distribution of atomic models across the processors on the server, and parameters for the model such as time and a description; and
- a zipped archive file containing the source code files, MA file, and event files.

To simplify development the DEVS Bridge provides capabilities to generate these files. A Model Information form allows the user to select the directory containing the DEVS model. The DEVS Bridge creates a zipped archive file containing the required files. Based on parsing the directory and the contents of the MA file the DEVS Bridge generates the XML files as well.

The DEVS Bridge uses HTTP messaging to create the DEVS model on the RISE server. Once the required files have been created, the DEVS Bridge uses the XML file to create the workspace for the model on the RISE server. The DEVS Bridge posts the files using the zipped archive file.

When the scenario completes, the DEVS Bridge deletes the model from the RISE server.

3.4 Use of C-BML Messaging Specification

3.4.1 Key elements of C-BML and their attributes

This architecture was developed to work as part of a larger synthetic environment where messaging between simulations is done via a message format defined using the C-BML standard. As described in section 2.1.4 C-BML was developed based on the JC3IEDM, a logical data model for use between different NATO countries. The model was developed with the aim of unambiguously conveying information required for planning and reporting on the operation of military organisations. It therefore contains elements designed to convey information about troops, equipment, terrain, installations, and weather conditions.

The JC3IEDM contains a number of independent entities from which all other entities are derived. Certain of these entities are re-used in the C-BML standard to convey Who, What and Where, three of the five W's that are meant to be addressed by C-BML messages. Four of the key entities are shown in the table below, along with their Definition, as defined in the JC3IEDM Main document, 2012. These types serve as the base types for elements in C-BML, as listed in the third column of the table.

Table 1 Independent Entities defined in the JC3IEDM Schema

Entity Name	Definition	Corresponding C-BML Type
Object-Item	An individually identified object that has military or civilian significance. Examples are a specific person, a specific item of materiel, a specific geographic feature, a specific coordination measure, or a specific unit.	AbstractObjectItem
Object-Type	An individually identified class of objects that has military or civilian significance. Examples are a type of person (e.g., by rank), a type of materiel (e.g., self-propelled howitzer), a type of facility (e.g., airfield), a type of feature (e.g., restricted fire area), or a type of organisation (e.g., armoured division).	AbstractObjectType
Location	A specification of position and geometry with respect to a specified horizontal frame of reference and a vertical distance measured from a specified datum. Examples are point, sequence of points, polygonal line, circle, rectangle, ellipse, fan area, polygonal area, sphere, block of space, and cone. LOCATION specifies both location and dimensionality.	AbstractLocation
Action	An activity, or the occurrence of an activity, that may utilise resources and may be focused	AbstractAction

	against an objective. Examples are operation order, operation plan, movement order, movement plan, fire order, fire plan, fire mission, close air support mission, logistics request, event (e.g., incoming unknown aircraft), or incident (e.g., enemy attack).	

All of the entities in the table are abstract types with extensive hierarchies that provide specialized definitions. Concrete types that can be used in messages are generally derived from a number of abstract classes, each of which defines attributes which apply to entities at that level. Thus the concrete elements have a large number of attributes, all of which need to be defined in order for the message format to be valid.

As an example, consider the Unit element. According to the C-BML schema, a Unit is defined as "a military ORGANISATION whose structure is prescribed by competent authority." The hierarchy of a unit is shown in the figure below.

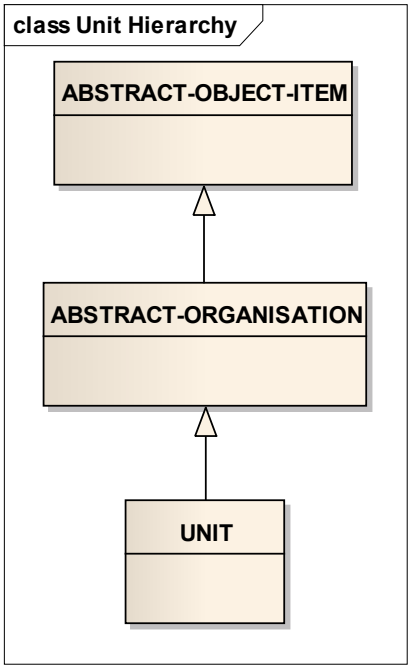

Figure 4 Unit Hierarchy

The complete set of attributes of a Unit and the level in the hierarchy at which they are defined is shown in the table below.

Table 2 Unit Attributes

Attribute	Description	Defined on
FormalAbbreviatedNameText	Character string specifying the common formal abbreviation used to designate a specific unit.	Unit
IdentificationText	The character string assigned to represent a unit's identification	Unit
OrganisationMaterielType-AssociationInOrganisation		AbstractOrganisation
OID	The globally unique object identifier. An OID can be any globally unique string (URL, GUID...)	AbstractObjectItem
NameText	The character string assigned to represent a specific OBJECT-ITEM	AbstractObjectItem
Alias	A child in a 'has' relationship	AbstractObjectItem
AliasRef	A child in a 'has' relationship	AbstractObjectItem
ObjectItemObjectTypeEstablishment-InObjectItem	A child in a 'is-assigned-establishment-through' relationship	AbstractObjectItem

In order for a single unit to be specified in any given C-BML message values must be supplied for all of these attributes.

The OID is not visible to users. However, it forms the basis of all relationships in the database. The OID assigned to a unit instance / object type is also the key used in all table entries that describe that

instance, e.g. Object-Item, Abstract-Object-Item, Organization-Item, Unit, Item, Location, and points making up the location.

The JC3IEDM breaks entity definitions into two parts: Object-Item and Object-Type. Type is a library of categories, e.g. Armoured Truck, four-door passenger car, Humvee. Item is an individual instance, e.g. the blue car license number XXXXXX, which is of type four-door passenger car. The Type library is filled in prior to the database being made available to commanders for planning.

When identifying an entity in C-BML, in addition to specifying the concrete sub-class of the AbstractObjectItem, a subclass of AbstractObjectType must also be specified. An Object-Item must always have an Object-Type associated with it. The Object-Type describes the category to which the entity belongs, e.g. type of role it performs, and the Object-Item describes specifics of the entity, e.g. name, location, current condition such as Damaged or Functional. The hierarchy of AbstractObjectType is similar to the hierarchy of AbstractObjectItem in that there may be many levels between the root type and the concrete types. Again, each level of the hierarchy defines a number of attributes. For example, the UnitType hierarchy is shown in the figure below.

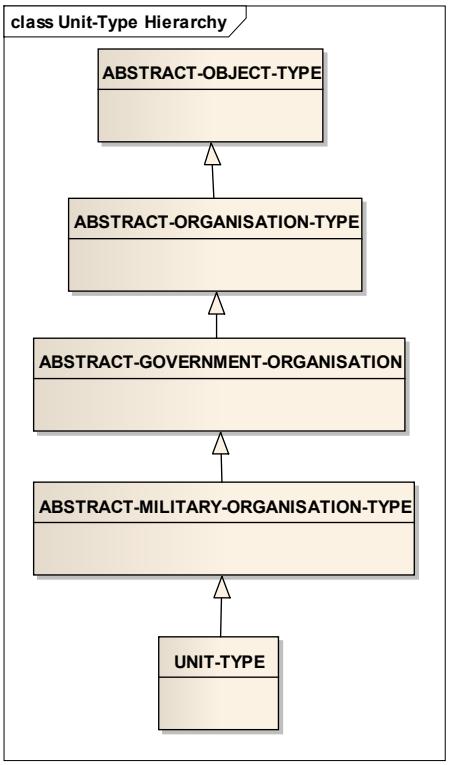

Figure 5 Unit Type hierarchy

Because of the complexity required to specify an entity, the C-BML specification also defines reference entities. The AbstractObjectItemRef and AbstractObjectTypeRef elements are parent classes of hierarchies of references that are used to associate items together. Most Reference elements contain only the OID, or unique id, of the referred-to item. All entities (Items, Types, Locations, Actions) have an OID. For example a Unit's location would be specified by using the LocationRef for that location. An Action that relates to a Unit would use the Unit's OID as a reference to that Unit.

The Action element similarly has a hierarchy with a number of concrete subclasses. The AbstractActionEvent element is used to describe activities of military significance that occur but for which planning is not known. The C-BML schema defines concrete types that are related to CBRN events: Chemical, Biological, Radiological or Nuclear.

All of this information is required to unambiguously convey information. However, parsing this information is not a trivial task. It has been noted in early experiments with the C-BML standard that parsing information requires multiple passes in order to be understood [16]. However, as noted in section 1.3, the goals of this thesis are to present an architecture for making DEVS models interoperable. The work required to properly parse and create fully-compliant C-BML messages is not among the stated goals. Since the DEVS Bridge is built using standard Web-enabled technologies, it is assumed for the purposes of this thesis that the DEVS Bridge can be extended to process C-BML messages. Therefore, full C-BML messages will not be used, either as input to the DEVS Bridge or as the input provided by the Bridge to the DEVS model.

3.4.2 Requirements and Non-Requirements for Scenario Execution

Instead of using full C-BML-compliant messages, a simplified message schema was developed. The message schema was developed to support the entities and messages defined in the Emergency Services Case Study, defined in section 5. The DEVS model represents emergency service units and a Dispatch model which tasks the units. The units are dispatched based on emergencies which are modelled by the external simulation. The external simulation supplies Reports of events to the DEVS model, which then responds by calculating the behaviour of its entities. The Reports need to include two of the five W's: What and Where. The format for Reports must therefore specify Action Events (What) and Locations (Where). The Who is not significant for reporting an event. The When is considered to be the start time of the scenario. Therefore, the Report schema need only describe the What and Where.

3.4.3 Custom Message Format and Attributes

The message format used for Report Messages in the scenario is shown below.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://tempuri.org/po.xsd"
xmlns="http://tempuri.org/po.xsd" elementFormDefault="qualified">

<xs:complexType name=”MessageBodyType”>
  <xs:sequence>
    <xs:element name=”ObjectId” type=”xs:string”/>
    <xs:element name=”CbmlMessageType” type=”xs:string”/>
    <xs:element name=”ActionEvent” type=”xs:string”/>
    <xs:element name=”LocationId” type=”xs:string”/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name=”MessageWrapperType”>
  <xs:sequence>
    <xs:element name=”OffsetTime” type=”xs:string”/>
    <xs:element name=”MessageLabel” type=”xs:string”/>
    <xs:element name=”MessageBody” type=”MessageBodyType”/>
    <xs:element name=”BodyType” type=”xs:string”/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name=”MessageListType”>
  <xs:sequence>
    <xs:element name=”MessageWrapper” type=”MessageWrapperType”
maxOccurs=”unbounded”/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name=”MSet”>
  <xs:element name=”MessageList” type=”MessageListType”/>
  <xs:element name=”ItemsAvailable” type=”xs:boolean”/>
</xs:complexType>
```

Figure 6 Report Message Schema

This figure shows the XML format of the messages, which is the format used for saving the messages to file. For communication with the DEVS model, a simplified comma-delimited string was selected as being better suited to the format used for input into the model. The comma – delimited format is shown below.

<ActionEventName,LocationId>

where

- ActionEventName is the string from the Action Event Category Code
- LocationId is the unique identifier for the location

3.5 Use of MSDL Model State Information Specification

3.5.1 Key Elements of MSDL and their attributes

The C-BML standard is used to define messages sent between simulations working together in a larger simulation, to communicate information about entities involved in the simulation. However, the exchange of messages is only meaningful if all of the simulations start with the same information about the entities. To achieve this, the Military Scenario Definition Language (MSDL) was defined. The MSDL schema is similar to the C-BML schema. All elements are identified by a Universally Unique Identifier (UUID), similar to the OID defined by C-BML.

The MSDL has a root element, Military Scenario. All elements required to define the scenario are defined under this element, grouped according to types. An MSDL file defines the Environment, including key locations, terrain and weather. It defines the people and organisations involved in the scenario, including organisations and force sides. It defines logical overlays used to group intelligence elements in the scenario.

3.5.2 Requirements and Non-Requirements for Scenario

The MSDL specification is designed to allow the definition of a large scenario with many entities. However, the case study selected for this experiment occurs on a much smaller scale with a small number of entities. For this reason the full MSDL schema is not required. A custom schema was defined that modelled just the entities required for the scenario.

3.5.3 Custom State Format and Attributes

Rather than defining a full-blown scenario, the custom schema defines a set of resources required for the scenario. The resource set contains two types of data: Locations and Units. The Locations are defined as part of the resources. This allows them to be referenced by their unique identifier in messages. Units are defined using the Location identifiers. The custom schema is shown in the figure below.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://tempuri.org/po.xsd"
xmlns="http://tempuri.org/po.xsd" elementFormDefault="qualified">
<xs:annotation>
  <xs:documentation>
    Format for saving resources
  </xs:documentation>
</xs:annotation>
  <xs:complexType name=”UnitDataType”>
    <xs:sequence>
      <xs:element name=”UnitName” type=”xs:string”/>
      <xs:element name=”UnitType” type=”xs:string”/>
      <xs:element name=”ActionTask” type=”xs:string”/>
      <xs:element name=”ObjectId” type=”xs:string”/>
      <xs:element name=”LocationId” type=”xs:string”/>
      <xs:element name=”HomeLocation” type=”xs:string”/>
      <xs:element name=”NextActivityTime” type=”xs:string” minOccurs=”0”/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name=”LocationDataType”>
    <xs:sequence>
      <xs:element name=”LocationName” type=”xs:string”/>
      <xs:element name=”Latitude” type=”xs:string”/>
      <xs:element name=”Longitude type=”xs:string”/>
      <xs:element name=”ObjectId” type=”xs:string”/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name=”LocationListType”>
    <xs:sequence>
      <xs:element name=”LocationData” type=”LocationDataType” minOccurs=”0”
maxOccurs=”unbounded”/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name=”UnitListType”>
    <xs:sequence>
      <xs:element name=”UnitData” type=”UnitDataType” minOccurs=”0”
maxOccurs=”unbounded”/>
    </xs:sequence>
  </xs:complexType>
<xs:complexType name=”ResourceSet”>
  <xs:sequence>
    <xs:element name=”UnitList” type=”UnitListType” minOccurs=”0”/>
    <xs:element name=”LocationList” type=”LocationListType” minOccurs=”0”/>
  </xs:sequence>
</xs:complexType>
```

Figure 7 Custom Model State Specification Format

When this data is sent to the DEVS Model, a simplified format is used. The format for Locations is shown below.

<LocationId,Latitude,Logitude>

where

- LocationId is the numeric unique identifier for the location. This is the value used in the event and units strings.
- Latitude is a double representing the latitude of the location
- Longitude is a double representing the longitude of the location.

The format for Units is shown below.

<UnitType,ActionTaskName,LocationId,HomeLocation>

where

- UnitType is the type of the unit
- ActionTaskName is the string from the Action Task Category Code
- LocationId is the unique identifier for the location
- HomeLocation is the location to which the unit returns when ordered to return to base

This string is repeated for each unit in the scenario. An example of the string is shown below:

<1000,Idle,100,100><2000,Idle,200,200>

3.6 Category Codes

3.6.1 JC3IEDM Category Codes

As described in the previous section, the JC3IEDM and C-BML standards define Category Codes to enumerate possible values for attributes of entities. Category Codes can be exchanged between different Command and Control (C2) systems in a language-independent way. The category codes are extensive, and are used across all types of entities. For example, there are multiple category codes that describe Organisation Types:

- Civilian Post Type Category Code
- Executive Military Organisation Type Category Code
- Group Organisation Type Category Code
- Private Sector Organisation Type Category Code
- Unit Type Category Code

The majority of the entity types defined in the schema require the use of one or more custom category codes. Action Events and Action Tasks have category codes, as do equipment types. While the codes

are extensive, most category codes also specify two special codes, Not Otherwise Specified (NOS) and Not Known (NKN) that can be used when the required value is not defined.

3.6.2 Issues with JC3IEDM Category Codes in Civilian Simulations

The extensive category codes and wide range of elements defined in the C-BML and MSDL schemas were developed for military organisations performing military operations. However, uses of the military often involve civilians. These may be local governments in an area where military operations are on-going. Military organisations also interact with civilian emergency organisations when responding to natural disasters such as flooding or fire [28]. Such joint interactions would benefit from the type of mission rehearsal and planning performed by the military. In order for civilian agencies to participate in large-scale simulations, the messaging format used would have to accommodate civilian agencies and the types of actions they perform.

The scenario selected to be modelled during validation of the architecture has civilian agencies responding to an emergency. C-BML was studied for use in defining the messages to be used. However, the category codes defined in the JC3IEDM specification do not seem to be suited to tasking civilian agencies the same way they support tasking military organisations.

The scenario described in section 5 Case Study – Emergency Services Dispatch requires units, locations, and tasks. The unit types are Police cars and Fire Trucks. The scenario starts with the units at their headquarters: Police station and Fire station. The units need to be tasked to fight a fire and perform crowd control.

A number of different category codes were examined for codes that would be useful for this scenario.

The Civilian Post Type Category Code defines POLCHF, the Chief Officer of a civil force to which is entrusted the duty of maintaining public order, enforcing regulations and detecting crime. This code allows the specification of the chief of police, but not regular police officers. There is no code for Fire Fighters.

The Military Post Type Category Code defines a number of different types of doctors, snipers and scouts, but not Military Police.

The Group Organisation Type Category Code defines POLCHF, Police Chief. It defines CRIMIN – Persons who attempt to profit by violating the law, and GANG. There are four different codes for media, and

codes for land owners, merchants, refugees, village elders and writers. However, there is no code for Police Officer or Fire Fighter.

The Private Sector Organisation Type Category Code defines business groups.

The Unit Type Category Code has no codes related to Police or Fire units. The Unit Type Arm Specialisation Code has a number of roles played by military units, including Military Police, but no Fire units.

According to the JC3IEDM Main document, the Action Event is defined as "…an ACTION that is an incident, phenomenon, or occasion of military significance which has occurred or is occurring but for which planning is not known" [14]. This entity is intended to capture ACTIONs that simply occur and need to be noted." The Action Event Category Code contains values for Arson, Accident, Bombing, Criminal Incident, Drug operation, Fire, Flood, Explosion, Murder, and a number of other types of crimes.

However, the Action Event Category Code also defines Firefighting, Patrolling, and Providing healthcare services. These values represent tasks that a commander may direct subordinates to perform. Action Event category codes cannot be used for tasking units.

The Action Task Activity Code is the code used in Abstract Action Tasks to identify the work to be done by the receiver of the Action Task. The code defines Air medical evacuation, Arrest/legal, Arrest/obstructing (stopping or checking motion, progress or growth), Capture, Command and Control, Evacuate, Medical evacuation, Patrol, Rescue, Search and Rescue, Secure an area, Provide protection, Search, and Support/contingency. These values support a wide range of tasks that need to be performed in emergency situations. However, there is no firefighting task.

3.6.3 Category Codes Required for Messaging and Entity State Data

While there is some support for the actions performed by civilian agencies, the scenario selected to validate the simulation interoperability architecture involves fire units fighting a fire, and there is no support for tasking fire fighters. For this reason, it was decided to create custom category codes for the simulation rather than try to re-use category codes defined in the C-BML schema. The use of custom codes is possible because of the use of custom messages and state description language.

3.6.4 Custom Category Codes

Three category codes were defined for the case study used to validate the architecture.

The Action Event category code represents observed actions that happen in the scenario. A small set of Action Events were defined based on the case study. These values represent not just the start of the event, but also report on the progress of the event. The codes are {FireInProgress, FireOut, EmergencyOver}.

The Action Task category code represents work that commanders instruct their subordinates to perform. These values are also used to represent the current state of the units in the simulation. The values selected were based on the case study used in validation of the architecture. In addition to the tasks to be performed, an Idle value was added to represent the state of the units at the beginning and the end of the scenario. The category codes are {Idle, Fight Fire, Clean Site, Crowd Control, Return to Base}.

The Unit Type category code represents the types of the simulated entities. It was restricted to only the two unit types required for the scenario modelled, Police and Fire.

3.7 Data Model

3.7.1 Location, Unit, Message

The Data Model represents both the Resource Set and the Message Set used in the simulation. The same model must be used by both the DEVS Bridge and the DEVS Model. This section discusses the Data Model that was used in the simulation used to validate the proposed architecture.

The Data Model consists of three classes:

- Locations
- Units
- Report Messages

The Location class describes locations of interest in the scenario. It is a simplified version of the Location class defined in C-BML. The C-BML standard defines a number of sub-types of locations, including Points, Ellipses, Fan Areas, etc. For this simulation a location is defined as a Point, with coordinates given as (Latitude, Longitude) rather than in MGRS format. The Locations defined in the Resource Set represent locations of interest in the scenario. These locations include the initial locations of the Units in the scenario, and the locations where the events take place in the scenario. All Locations that are

required in the scenario are defined in the Scenario Definition files that are loaded during the initialization of the scenario.

The attributes of the Location Class are:

- Unique ID – used to reference the location in messages from external simulations and by the Unit class.
- Latitude – Latitude in decimal format
- Longitude – Longitude in decimal format

The Unit Class describes entities that participate in a scenario. It is a simplified version of the Unit-Object-Item defined by the C-BML standard. Attributes of the Unit Class are:

- Unique ID – used to reference the unit in messages from external simulations.
- Current Action Task – current activity being performed by the Unit. This value is taken from the custom Action Task Category Code as described in section 3.6.
- Location – Unique ID of the Location object which represents the current location of the Unit.
- Home Location – Unique ID of the Location object which represents the location to which the Unit returns at the end of the scenario.
- Unit Name – Name of the Unit.
- Unit Type – Type of the unit. The value from the Unit Type Category Code defined for this thesis as described in section 3.6.

The Unit Type field replaces the Unit-Object-Type defined by the C-BML standard. For the purposes of validating the architecture, detailed object type information is not required.

The DEVS Model represents Units as Atomic Models. Each time the DEVS Model executes, it calculates:

- a new Action Task for the Unit to execute,
- a new Location for the Unit; and
- the time it will take for the Unit to reach that Location.

Since this calculation may depend on the previous Action Task and Location of the Unit, the previous values are loaded into the atomic models during the initialization of the DEVS Model prior to its execution. The newly-calculated values are produced as output from the DEVS Model and parsed by the DEVS Bridge. The DEVS Bridge updates its internal Data Model with this value. The next time the DEVS

Model must be executed, the DEVS Bridge uses its internal Data Model to generate the initial Scenario Description file for the DEVS Model.

Report Messages represent the messages in the script run by the DEVS Bridge. They are simplified versions of the C-BML Report message. Messages have two functions in the execution of a scenario: they deliver new information, such as an Action Event or an Order, and they trigger the execution of the DEVS Model to determine the response of its simulated entities. The attributes of Report Messages are:

- Report ID – unique identifier for the Report Message
- Action Event – a value from the Action Event Category Code described in section 3.6.
- Location – Unique ID of the Location at which the Action Event takes place.

3.7.2 Relationships between Model Entities

The relationships between the entities in the model are shown in the figure below.

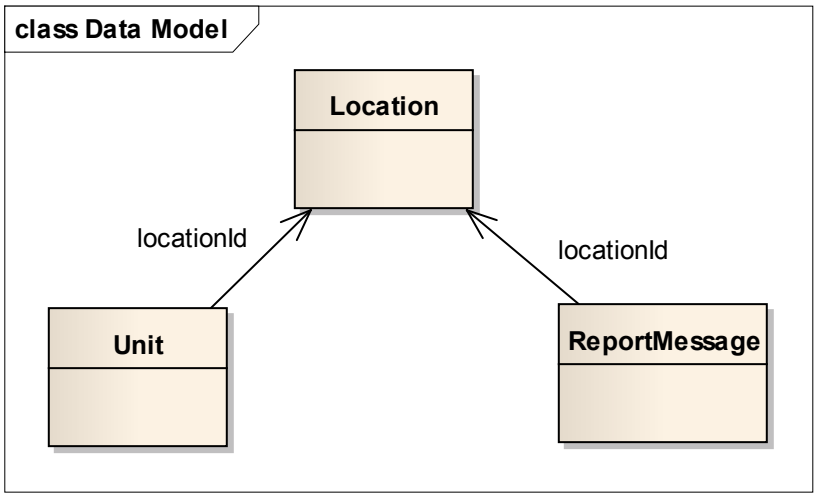

Figure 8 Data Model Class Diagram

The Unit class has attributes that use the Action Task and Unit Type category codes to identify its current activity and its type. The Report Message uses the Action Event category code to identify the activity being reported at the Location associated with the Report Message.

3.7.3 Location Calculations in the Scenario

The DEVS Model Location class calculates the time required to move from one location to another. Since the scenario being implemented is a simplified scenario, the time calculation is also simplified. The distance is calculated assuming a Manhattan layout of a city, with the two locations forming diagonally opposite corners of a box. The distance travelled is the length of two sides of the box, as shown in Figure 9 below.

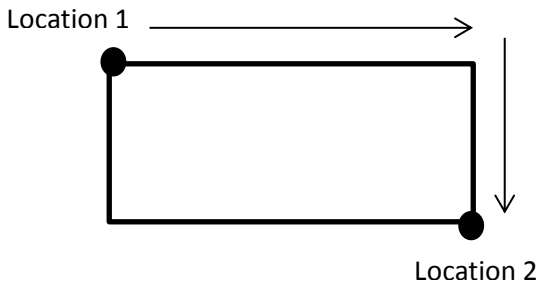

Figure 9 Distance Calculation

The distance is multiplied by speed, which is passed into the location for the purpose of calculation. The speed is an attribute of the Unit that requests the calculation. This attribute is unique to the DEVS Model implementation of the Unit model.

4 Implementing the Architecture – DEVS Bridge and DEVS Model

4.1 DEVS Bridge Implementation

4.1.1 Main Interface

The DEVS Bridge was implemented to validate the proposed architecture for adding a simulation interoperability interface to models on the DEVS server. The DEVS Bridge was implemented as a stand-alone application that uses HTTP to communicate with other systems. It implements the scripting capability described in section 3.3.1 along with an interface that can be used to execute the messages in the script. It implements the capability to create the files needed to post a DEVS model to the RISE server, and the capability to post and execute the model. It implements the capability to maintain the current state of the modelled entities and generate the Report messages and state information required by the DEVS model to execute. The following sections discuss the implementation in more detail.

The capabilities of the DEVS Bridge are organized into components. The components are:

- the DEVS Bridge main component – implements the majority of the capabilities, as described below.
- the Model Information component – implements the capabilities related to defining the DEVS model on the RISE server and preparing the model files for posting on the RISE server,
- the Message Set component – implements the capabilities for defining the script to be executed; and
- the Resources Set window – implements the capabilities for defining the set of simulated entities and other resources such as locations that are relevant to the simulation to be executed.

The responsibilities of each of the components are described in the sections below.

DEVS Bridge Component

The DEVS Bridge component has the following capabilities:

1. It loads the Message and Resource Set files required for the scenario.
2. It maintains the current state of the entities in the scenario (units, locations) between executing messages in the scenario.
3. It generates the Report Message in the DEVS-native format.

4. It generates the current model state description in the DEVS-native format.
5. It interacts with the RISE server to post and execute the DEVS model.
6. It steps through the set of messages to execute the scenario; and
7. It displays status information and response data from messages sent to the RISE server.

Model Information Component

The Model Information component has the following capabilities:

1. It allows the user to select the DEVS Model files to be used in the simulation.
2. It generates the XML file used to create the workspace on the RISE server; and
3. It creates the zipped archive file used to post the model to the RISE server.

Resource Set Component

The Resource Set Component has the following capabilities:

1. It allows the user to create and edit the set of resources used in the scenario.
2. It allows the user to save the Resource Set to a file; and
3. It allows the user to load the Resource Set from a file.

Message Set Component

The Message Set Component has the following capabilities:

1. It allows the user to create and edit the set of messages used in the scenario.
2. It allows the user to save the Message Set to a file; and
3. It allows the user to load the Message Set from a file.

4.1.2 Script Definition and Management

The Message Set and Resource Set are maintained in memory by the DEVS Bridge during the execution of the scenario. These classes implement the Data Model described in section 3.7. In addition to the classes that implement this model, there are also classes that group the resources and format messages to send to the RISE server. The diagram for the data classes is shown in the figure below.

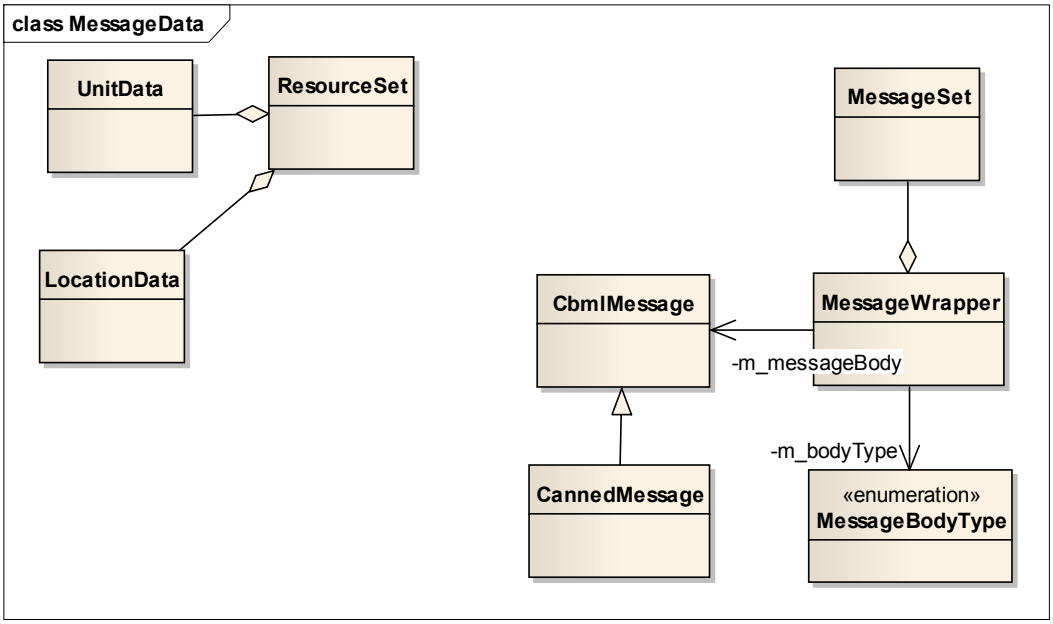

Figure 10 Message Classes

The classes in the figure are:

1. UnitData – This class implements the Unit class defined in the logical Data Model described in section 3.7.
2. LocationData – This class implements the Location class defined in the logical Data Model described in section 3.7.
3. ResourceSet – This class implements the set of all resources required in a scenario. It has a list of UnitData instances and a list of LocationData instances.
4. CbmlMessage – This class is used to create the messages in the scenario script. It implements the Report Message described in section 3.4. It has the action event and location id as attributes. It also implements the capability to serialize the message as both an XML message and the DEVS Model native format described in section 3.4.
5. CannedMessage – This class is used to create messages where the message body format is known exactly. It is used to specify the Message Body for messages that are sent to the server to create, execute and get the results from the DEVS Model.
6. MessageWrapper – This class is used to define Http Messages. It has a message body attribute that is a CbmlMessage instance, as well as attributes for managing its use in the script, including a label string for display purposes. It contains the logic for serializing the message. It also

includes the logic for loading the message body into the Http Message depending on the message type – zipped file, xml or text.

7. MessageSet – This class represents the script for a scenario. It contains the set of MessageWrappers with the CbmlMessages to be executed during a scenario. It has the logic for serializing the message set so that it can be saved to file and read from file. It also has an attribute that represents the type of HTTP message is to be used to send the CbmlMessage to the server.
8. MessageBodyType – This is an enumerated value that indicates the type of the message body for HTTP message: Zipped file, text, no message body.

4.1.3 Model Creation on RISE Server

During execution of the scenario, the DEVS Model must be created and posted to the RISE server. This involves a number of steps:

1. Create the workspace on the RISE server.
2. Post the model to the server.
3. Execute the model; and
4. Obtain the results.

The sequence of events is similar for all of the listed steps. The differences between the steps are the URL employed in the HTTP message, the RESTful message type and the Message Body. The list of message types and the message bodies are shown in the table below. The URI column shows the end of the URI that is appended to the address of the main workspace in the server, e.g. http://134.117.53.66:8080/cdpp/sim/workspaces/elizabeths/dcdpp/model_name/

Table 3 URI Components

Activity	URI Ending	RESTful message type	Message Body
Create Workspace	n/a	Put	XML file describing the model
Post the model files	?zdir=<modelName>	Post	Zipped archive file
Execute the model	/simulation	Put	None
Get Results	/results	Get	None

Delete the model	n/a	Delete	None

Once the contents of the message have been identified, the same sequence of steps is followed. This sequence is shown in the figure below.

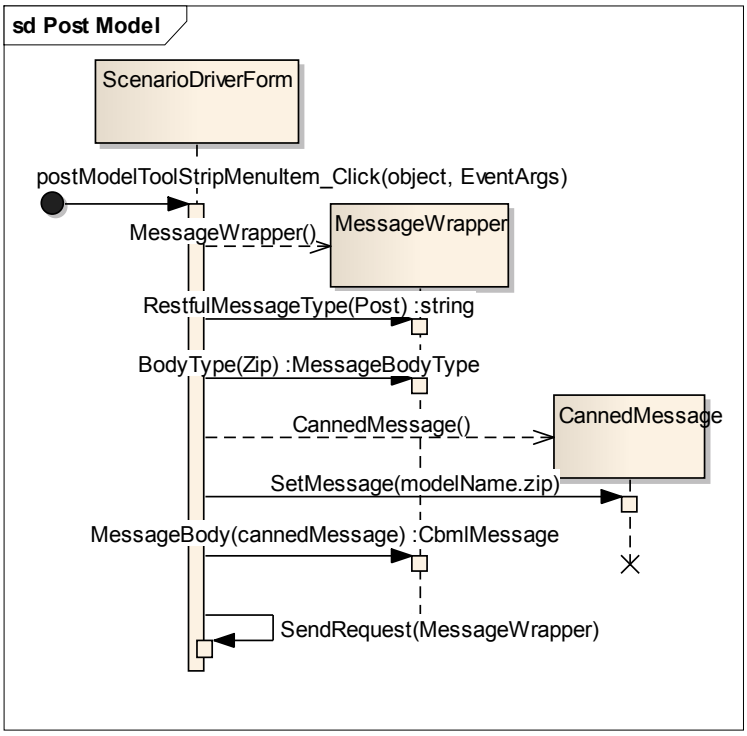

Figure 11 Post Model Sequence Diagram

The sequence is triggered by the user selecting a menu option from the DEVS Bridge window. The steps in the sequence are described below:

1. postModelToolStripMenuItem_Click – User invokes the capability to the post the model. The ScenarioDriverForm is the component that implements this capability. This method executes the rest of the steps in this sequence. The XML model-definition file and the zipped archive file must already have been created before this method is invoked.
2. MessageWrapper() – A new instance of the MessageWrapper is created via its constructor.
3. RestfulMessageType(Post) – The RESTful message type, POST, is set on the MessageWrapper.
4. BodyType(Zip) – The HTTP Message body type is set. For the operation that posts the model to the RISE server, the HTTP Message type is set to Zip.
5. CannedMessage() – A new instance of a CannedMessage is created;

6. SetMessage() – The text of the Canned Message is set with the text in the URI Ending column in Table 3 URI Components.
7. MessageBody(cannedMessage) – The instance of the CannedMessage is saved in the MessageWrapper.
8. SendRequest(MessageWrapper) – The message is sent to the RISE server.

The behaviour of the system in the SendRequest method is shown in the figure below.

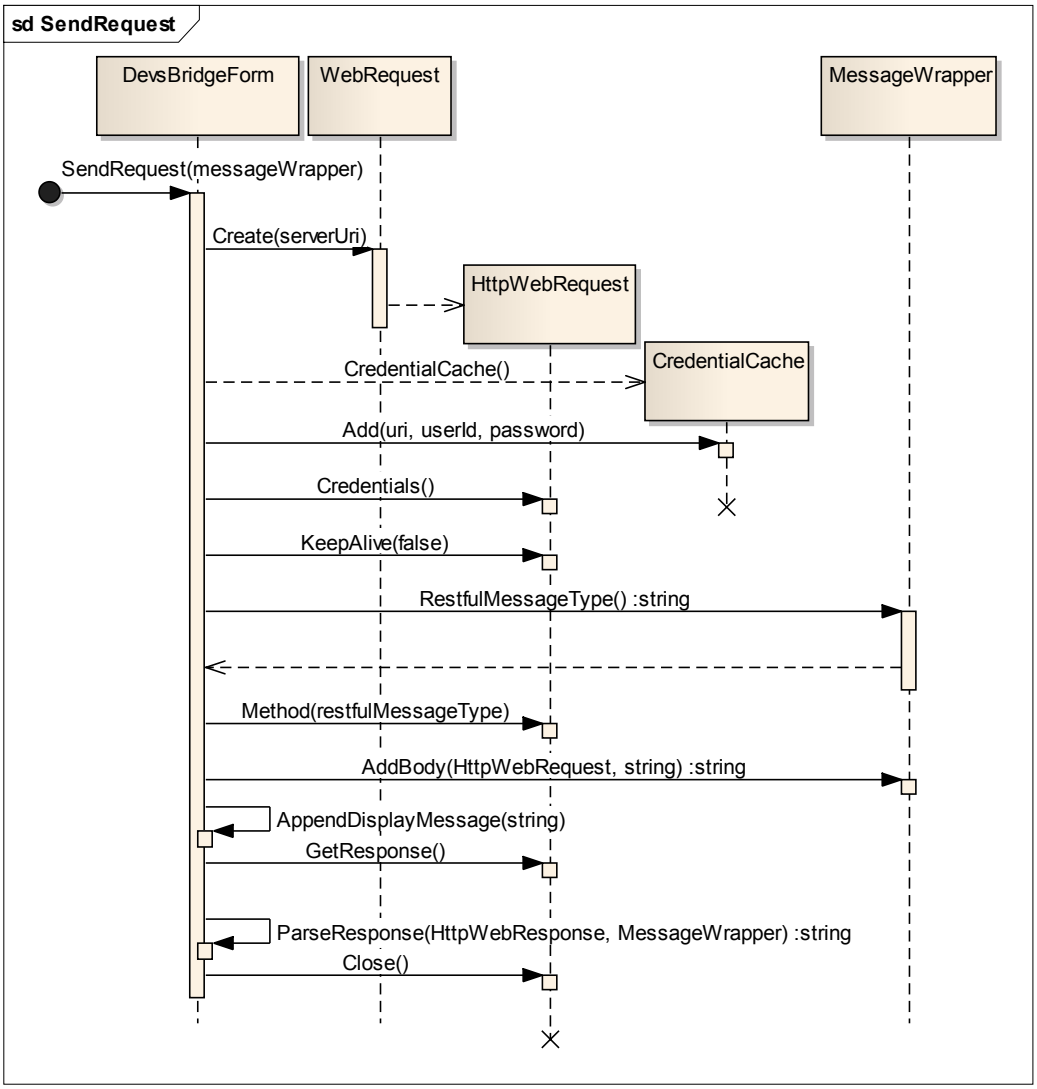

Figure 12 Send Rquest Sequence Diagram

The SendRequest sequence is invoked for all messages sent to the RISE server. The steps in the sequence are described below.

1. SendRequest(messageWrapper) – The method is invoked after the attributes of the messageWrapper have been populated with the correct attributes.
2. Create(serverUri) – This is a static method called on the class. It creates an instance of the HttpWebRequest class initialized with the full URI for the RISE server.
3. CredentialCache() – An instance is CredentialCache is created. This class contains the login values required to access the server.
4. Add() – Adds the name and password values to the instance of CredentialCache.
5. Credentials() – Saves the instance of the CredentialCache on the HttpWebRequest.
6. KeepAlive(false) – Sets an attribute that indicates that the communication with the RISE server should not be kept open after the request is handled.
7. RestfulMessageType() – The type of message (PUT/POST/GET/DELETE) is extracted from the MessageWrapper instance.
8. Method() – Sets the RESTful message type on the HttpWebRequest with the value from the MessageWrapper.
9. AddBody() – The MessageWrapper is invoked to set the message body of the HttpWebRequest. For zipped message bodies, the entire zipped archive file is loaded. For an XML message the XML file contents are loaded.
10. AppendiDisplayMessage() – The contents of the message are displayed to the user to indicate the progress of the request.
11. GetResponse() – This method connects to the RISE server via the URI, passes the message body, and waits for a response.
12. ParseResponse() – The response returned from the server is parsed. If it contains a zipped archive file with results, as is the case when getting the results of execution, the file is unzipped and saved to disk. A message is displayed to the user indicating the results returned from the server.
13. Close() – Closes the connection to the server.

4.1.4 Messaging/Response with DEVS Model

When the DEVS Model runs it requires two sources of input: the Report Message and the current state description. The Report Message is generated by the DEVS Bridge based on the current message in the

script. The state description of the atomic models is generated from the data model currently held by the DEVS Bridge. These activities are shown in the figure below.

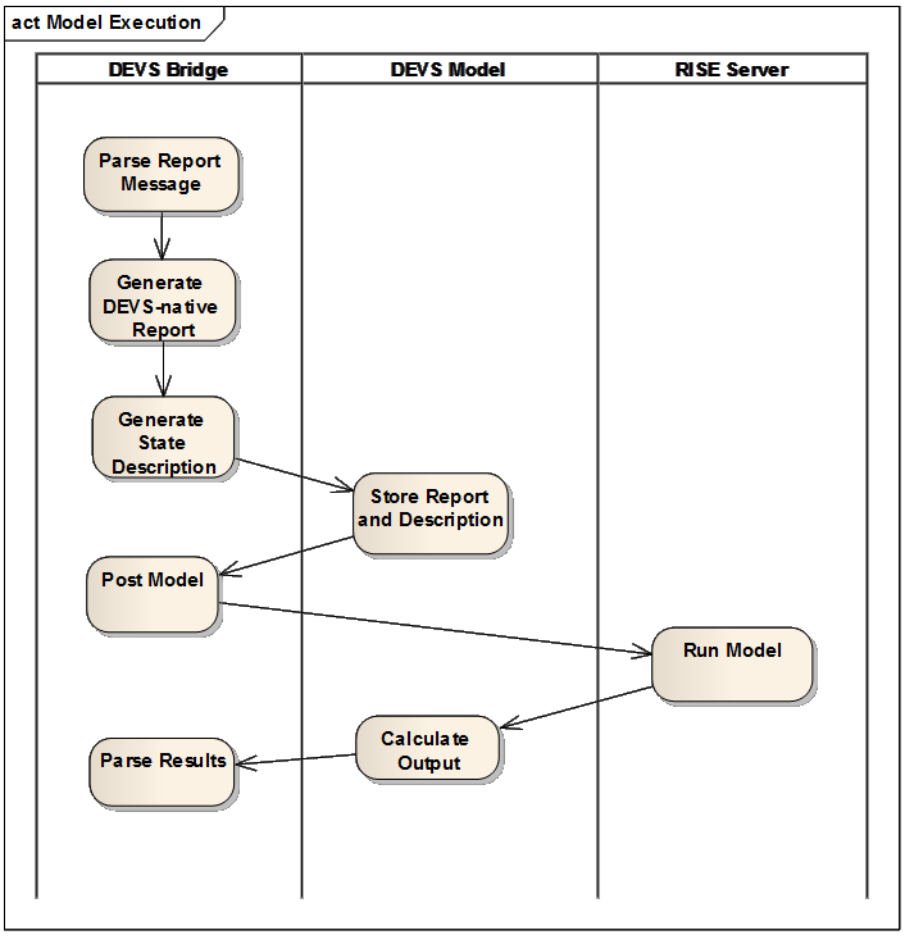

Figure 13 DEVS Model Execution Activity Diagram

The activities in the diagram are:

1. Parse Report Message – The DEVS Bridge interprets the message sent from an external simulation. In the case of this experiment this is done by loading the next message in the script.
2. Generate DEVS-native Report – The DEVS Bridge takes the parsed-in Report Message and creates a Report message in the format used by the DEVS Model. This is the format described in section 3.4.

3. Generate State Description – The DEVS Bridge uses the internal state of the simulated entities and creates the State Description in the DEVS native format. This is the format described in section 3.5.3.
4. Store Report and Description – The Report and State Description are saved with the files that make up the DEVS model.
5. Post Model – The DEVS Bridge zips the model and generates the XML file that describes the model, creates the workspace on the RISE server, and posts the model to the RISE server.
6. Run Model – In response to an invocation by the DEVS Bridge, the RISE server initializes and runs the DEVS model.
7. Calculate Output – The DEVS model runs and outputs the next state of the simulated entities.
8. Parse Results – The DEVS Bridge retrieves the output from the execution of the DEVS model and parses the results to update its internal data model.

4.2 DEVS Model Discussion

4.2.1 Requirements on DEVS models

The DEVS Bridge provides the link between the external simulations and the RISE server. However, in order for a DEVS Model to be leveraged by the DEVS Bridge it must meet a number of criteria.

1. It must define an atomic model that functions as a Gateway.
2. All atomic models must load their initial state data as part of their initialization function.
3. Unless the DEVS model type is real-time DEVS, the model as whole must perform as a calculation engine, rather than an end-to-end simulator.

These requirements are described in more detail below.

4.2.2 Role of the Gateway Model

The DEVS Bridge acts as an interpreter between the rest of the synthetic environment and the DEVS Model. It parses incoming C-BML messages and maps them into a format that is easier to parse by the DEVS Model. However, this format represents another coupling between the DEVS Model and the DEVS Bridge. Any of the Atomic Models that require the information in the incoming Report Message must be able to understand the format being used. Any changes to this format require corresponding changes to the Atomic Models, thus increasing maintenance efforts. During the preparation phase of an exercise

using C-BML the message format details may change as the requirements are finalized [20]. Changes in the details of the external messages may require changes to the format of the messages recognized by the DEVS Model. Reducing the number of Atomic Models that parse the DEVS Model Report Message formats reduces maintenance required.

MSDL, the scenario description language, and the custom state description language used by the DEVS Model are based on the messaging schemas used in the synthetic environment. They also may be affected by changes to the official schema descriptions for the simulation as the descriptions evolve. Again, maintenance of the DEVS Model is improved if the Atomic Models do not need to be able to parse these messages.

For these reasons, the first guideline for developing DEVS Models is that it must contain an Atomic Model that performs a Gateway function. This class has three main responsibilities:

1. Parse the state description information and save it in objects representing the data model that are available to all atomic models,
2. Parse the Report Message. The contents of the message may be stored in the data model objects or mapped to an event that can be used to trigger the start of processing of the DEVS Model; and
3. Start the simulation by sending the first event to another model.

The Gateway must parse the Report Message and initial state description information so that the other atomic models can use the data without having to parse it first. This mapping must be completed before the DEVS Model begins processing. Once parsed in, the data must be saved in objects that implement the data model. It may be stored as static variables on the Gateway class, or as external variables. Another method may be to have the scenario start with the Gateway class sending events to the individual atomic models in the scenario containing the initial data for the model.

All initialization of the data model must be complete before the individual atomic models initialize, so that the data is available to the atomic models. After all of the atomic models are initialized the Gateway can start the execution of the model by sending an event representing the receipt of the Report Message to a top level model. Once this is complete, the Gateway model should not participate any further in the execution of the scenario, unless it is to supply data model data to a processing atomic model. The exception to this may be if the data model objects are attributes of the Gateway model.

4.2.3 Data Model and Its Use By Atomic Models

The data model details will depend on the scenario being modelled. However, there are some general guidelines that must be followed.

The initialization of the atomic models must be done before the models begin processing the Report Message, but after the Gateway has parsed in the initial state description. This may require coordination between the models to ensure the timing. One possible technique is to have the Gateway perform its work during initialization, and then have the atomic models schedule internal transitions two or three milliseconds into the scenario. The atomic models then perform their initialization during the internal transition.

4.2.4 Behaviour Requirements on Atomic Models that work with the DEVS Bridge

The DEVS Model in the proposed architecture is being used to calculate the next state of simulated entities. Normally the DEVS Model would simulate the behaviour of the simulation entities from the start of the scenario all the way through to the end of the scenario. However, for this scenario, the model needs to be designed so that it only simulates the behaviour of the entities one step at a time, where one step is the behaviour of the entities from the receipt of a message from an external simulation until the entities will make no further state changes without receiving another external message.

The DEVS Models must also output any information about the new state of the simulated entities that needs to be sent to the external simulation(s). As an example, consider a scenario where the DEVS Model simulates a warehouse with inventory and entities A, B and C which represent delivery trucks. The larger synthetic environment represents a factory that uses just-in-time delivery of resources in the manufacture of cars. The factory is modeled by another simulation system, which also controls the execution of the simulation. When the factory requires parts held in the warehouse modeled by the DEVS Model, it sends a request. The DEVS Model receives the request, determines whether it has enough parts to fill the order, and then assigns a truck to deliver the parts. The truck to perform the delivery is selected based on the current locations of the trucks when the request is received. The input to the DEVS Model is the location of the trucks when the order is received, the current inventory of the warehouse, and the request message. The output from the DEVS model is the truck that services the request, and the time at which the truck will arrive at the factory. The truck id and time must be output from the DEVS Model so that it can be parsed by the DEVS Bridge and supplied to the factory simulation.

5 Case Study – Emergency Services Dispatch

5.1 Scenario Overview

5.1.1 Civilian Emergency Services Planning

For the purposes of validating the architecture, a DEVS model was required. The selected scenario models the command and control of Emergency Services. As discussed in section 2.2.5 civilian emergency agencies are looking at planning their response to large scale situations such as fires, floods, accidents and terrorist activities such as bombings. In some of these scenarios, such as natural disasters or bombings, civilian agencies may need to interact with military organisations for the sake of additional manpower or the specialized skills of the military.

In order to perform joint planning exercises, civilian agencies need to be able to model their resources, as well as their capacity to respond to emergencies. Civilian agencies operate in response to reports of an emergency event that is not planned ahead of time. Those in command of the agency may decide to dispatch additional resources as the situation continues or escalates, or recall trucks or units as the situation comes under control. This event-response nature of situation management makes the scenario suited to modelling as using a Discrete Event simulation.

5.1.2 Real World Entities

In real-world scenarios when an emergency occurs, such as a fire, observers call a central number such as 911 and report seeing the fire. In response the 911 dispatcher contacts the nearest fire department and orders the fire crews to the site of the emergency. The dispatcher will also send police cars to the scene to provide support, such as crowd control or to secure an area. If the fire is too large, the fire fighters on the scene may request that the dispatcher send more fire trucks to the scene.

After the flames have been put out, the fire fighters may stay on the scene to ensure that there are no burning embers that may start a new fire. Once the fire fighters are satisfied that the scene is under control the fire trucks on the scene are directed to return to their stations.

The entities in the system and their relationship are shown in the figure below.

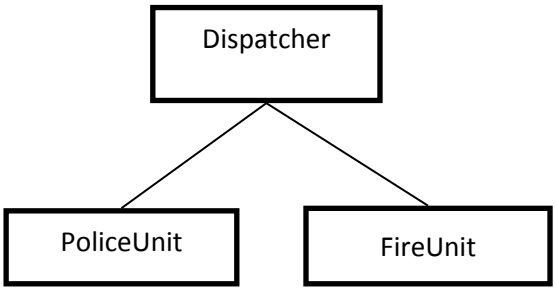

Figure 14 Conceptual Entities in the Emergency Services Dispatch scenario

The Police and Fire departments have their own Dispatch services, but for the sake of this scenario only a single Dispatch entity is modelled.

5.1.3 Activities

The scenario that was modelled for this case study is a simplified model of the activities that might occur during the response of emergency services to a fire. The steps involved are:

- A fire is reported to emergency services, possibly by dialing a central number such as 911 in North America.
- Fire and Police units are sent to the scene of the fire.
- The fire is put out, but the units remain on scene to ensure the fire is out.
- The emergency is determined to be over, and the units are recalled.

The activities that take place are shown in more details in the diagram below.

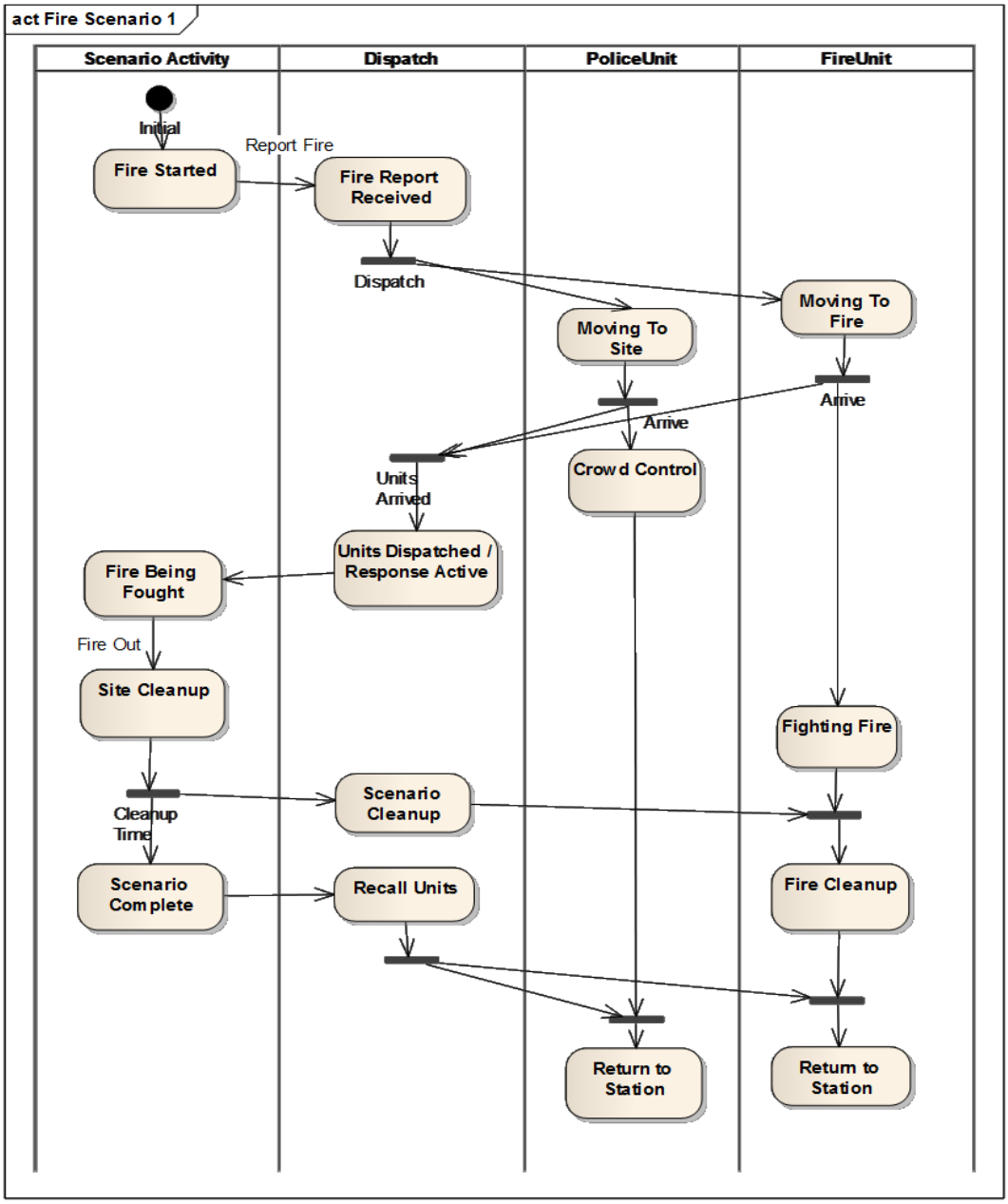

Figure 15 Conceptual Activity Sequence in a basic scenario

The activities are broken up according to the entity that performs them. Activities in the swim lane labeled “Scenario Activity” describe the overall activities at the site of the emergency. The activities in the other swim lane represent activities that must be performed by modelled entities.

The actual behaviours of the real entities in the simulation are more complex. However, as this model was developed to validate the proposed architecture, the main activity performed by the models of the

entities are restricted to calculating the time at which the entities arrive at a new location and start to perform a new task.

The individual activities in the scenario are:

1. Fire Started – The external simulation starts and sends a report that a fire has started. The report includes the location of the fire.
2. Fire Report Received – The Dispatch unit receives the report. Because this is a fire, the Dispatch operator determines that both Police and Fire Units are required at the scene.
3. Moving To Site (PoliceUnit) – The Police Car which is tasked to respond, either because it is closest to the site or because it is not on another call, begins heading to the site of the fire.
4. Moving To Site (FireUnit) – The Fire Truck, or Fire Company (all the trucks at a given Fire Station) begins heading to the site of the fire.
5. Arrive (PoliceUnit) – The Police Car arrives at the site and begins to perform Crowd Control, possibly by setting up barriers to keep bystanders away from the fire. It reports back to Dispatch to indicate that it has arrived.
6. Arrive (FireUnit) – The Fire Truck(s) arrive at the site and begin fighting the fire. They report back to Dispatch to indicate that they have arrived on site.
7. Units Dispatched / Response Active – The Dispatch operator notes that the responders have arrived at the scene of the fire.
8. Fire Being Fought – The overall state of the scenario changes to reflect the fact that the fire is being fought.
9. Crowd Control – The Police officers engage in Crowd Control.
10. Fighting Fire – The Fire Fighters attempt to extinguish the fire.
11. Site Cleanup – Due to the activities of the Fire Fighters the flames are extinguished.
12. Scenario Cleanup – The fact that the fire is out is reported to the Dispatch officer, who notes that the activities at the site have changed.
13. Fire Cleanup – The Fire Fighters change activities from fighting open flames to ensuring that there are no hot spots or burning embers, and possibly begin looking for the source of the fire.
14. Scenario Complete – The responders on the site of the fire determine that no further action is required. This fact is reported to the Dispatch officer.
15. Recall Units – The Dispatch officer sends a message to the units on the scene that they can return to their respective stations.

16. Return to Station (Police Unit) – The Police Car returns to the Police station, or otherwise resumes its routine activities.
17. Return to Station (Fire Unit) – The Fire Truck(s) return to the Fire station.

These behaviours formed the basis for the creation of the DEVS Model that was developed to validate the DEVS Bridge Architecture. The details of the model are described in the section below.

5.1.4 Integration of the DEVS Model with the Synthetic Environment

The Synthetic Environment is made up of distinct simulation services. Each manages the behaviour of one or more entities that participate in the larger simulation. They broadcast the behaviour of the entities they manage. Other simulations subscribe to these messages and modify the behaviour of their entities in response.

As an example, the larger synthetic environment for the case study may consist of four separate simulations. Model A simulates the behaviour of a building which catches fire. Model B is a DEVS model which simulates people evacuating the building as the fire breaks out. Model C simulates the behavior of people on the street around the building using Agents. Model D is the dispatch system from the Case Study.

Model A acts as the simulation controller and starts the fire. It sends out a message indicating that the fire has started. This causes Model B to start evacuation, with the evacuation paths being affected by the location in the building of the flames. When Model C receives the Fire Started message, some civilians begin to move away from the building, while others may come closer to watch. One or more civilians place calls to 911 emergency services. Model D receives the emergency services notification and dispatches Fire and Police units to the location of the fire. On arrival at the scene, Model D sends a message indicating that the Fire units have started to fight the fire. Model A receives this message. After a period of time based on the size of the fire and the duration of the Fire units applying water, Model A determines that the flames have been put out. Model A sends a message indicating that the flames are out. The Dispatcher service in Model D instructs the Fire Units to begin clean-up. The people modelled by Model C begin to leave. When the fire is out Model D recalls the units.

5.2 DEVS Model Overview

The Emergency Response scenario described above was implemented to validate the DEVS Bridge architecture. It was implemented using DCD++, the distributed CD++ tool, using C++.

The Emergency Response services model that evolved from the development of the scenario in the previous section has three main models. Assuming that the activities in the Scenario Controller column are performed by the external simulation, the DEVS Model requires a Dispatcher, Police Unit and Fire Unit. As described in section 4.2.2 a Gateway model is also required. The resulting set of entities and their relationships are shown in the figure below.

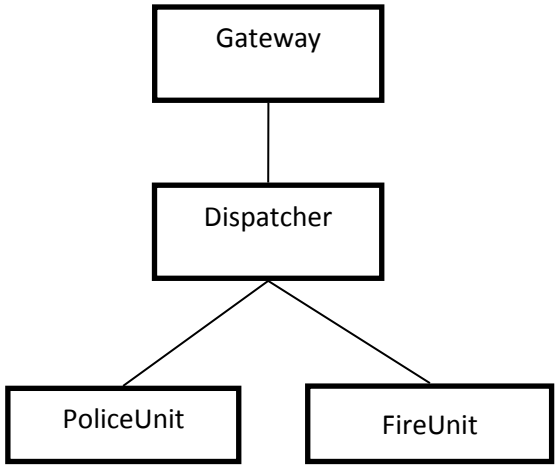

Figure 16 Conceptual Entities in the Emergency Services Dispatch scenario

The entities described in the figure become the atomic models in the DEVS Model of the Emergency Services, which is used to validate the architecture proposed in this thesis.

5.2.1 DEVS Model

The four atomic models are arranged with three of them arranged into a coupled model named Services. The relationships between the models are shown in the figure below.

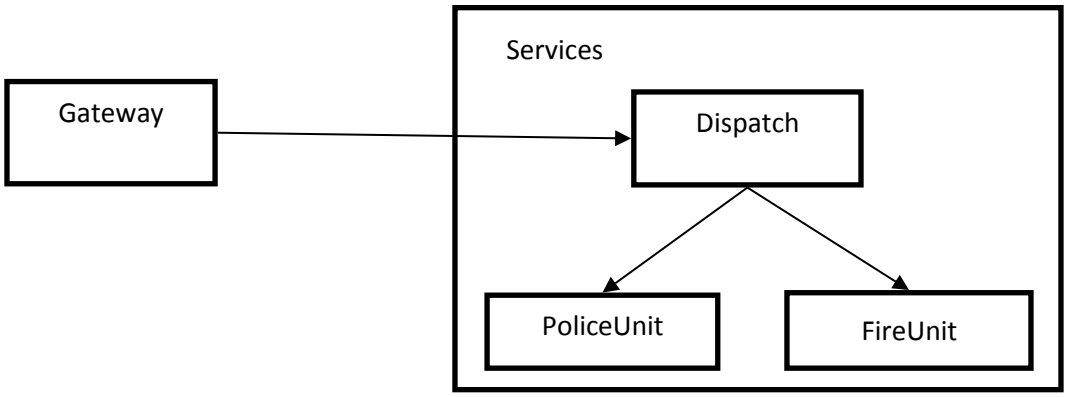

Figure 17 DEVS Models

Each of the arrows showing connections between atomic models represents two port connections, one that passes Action Event information and another that passes the Location ID of the location where the Action Event takes place.

5.2.2 Role of the Gateway Model

As described, the role of the Gateway atomic model is to coordinate the behaviour of the models in the simulation. During its initialization it parses the initial model state information into the internal data model, which is represented by the Unit, Location and MessageHolder classes The MessageHolder class contains a list of Locations and Units which are populated by parsing the state description information. The MessageHolder class also contains the Report Message. The MessageHolder instance is populated by the Gateway model. The classes are shown in the figure below.

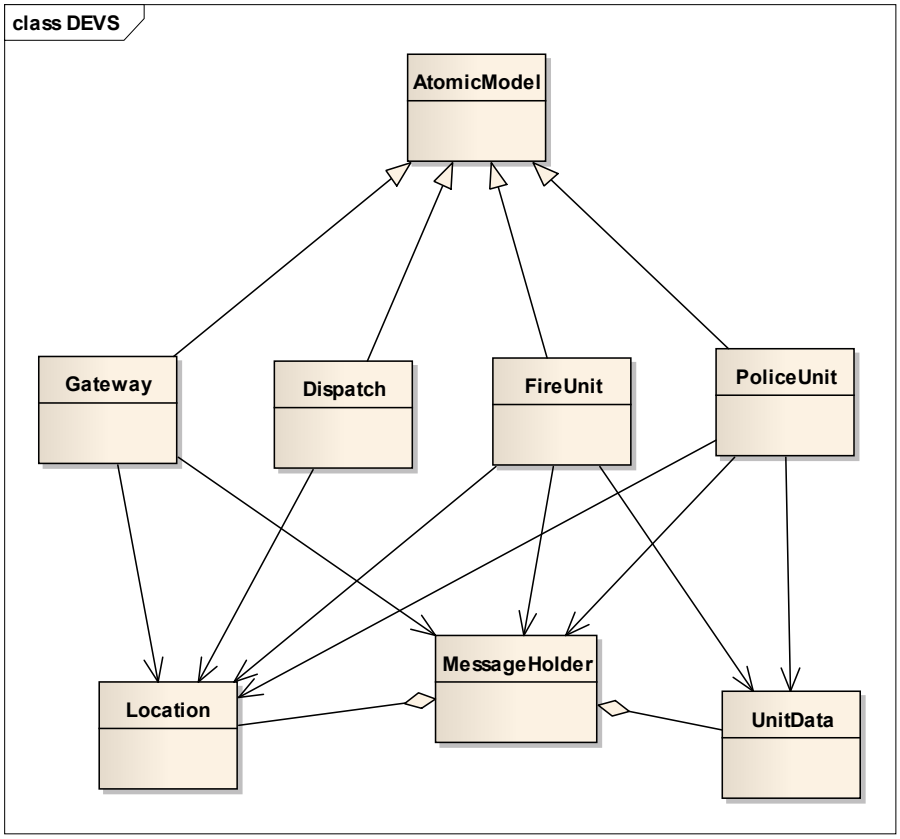

Figure 18 DEVS Model Class Diagram

The MessageHolder class instance is defined on the Gateway implementation file. The other Atomic Models access the MessageHolder by declaring external definitions for the instance. In this way the list of locations and initial unit state data is accessible to the other atomic models during the execution of the scenario.

The Report Message and initial state description information are included in the MA file for the model as two separate parameters. These values are loaded and parsed by the Gateway in the initialization function of the Gateway model. This allows the parsed data to be available to the Police and Fire unit internal transitions that load the initial data. See the section below on the PoliceUnit and FireUnit for more information.

5.2.3 Connections between Atomic and Coupled Model

The Gateway model is connected to external ports. Events are scheduled on these ports. The events are scheduled after the models have all initialized. When the Gateway receives external events, it sends

the Action Event from the Report Message to the Dispatcher along with the Location ID from the Report Message.

As shown in Figure 17, at the top level the Gateway atomic model connects to the Services coupled model. The ports on the Service model that connect to the Gateway also connect to the Dispatch model. Thus the output Action Event and Location ID from the Gateway are passed to the Dispatch model. The Dispatch Model is connected to the PoliceUnit and FireUnit models. The arrow representing the connection between the Dispatch Model and the Unit models represents two ports each, one for the Location ID and one for the Action Event.

The PoliceUnit and FireUnit are connected to external ports on the Services model, which are connected to external ports on the top model. When the PoliceUnit and FireUnit export their results, the results are exported from the overall model and appear in the out file.

5.2.4 Responsibilities of the Dispatch Model

The Dispatcher entity is responsible for receiving reports from the scene of the emergency and assigning tasks to the Police and Fire Units. The tasks it assigns depend upon the report received. The breakout of these tasks is shown in the table below.

Table 4 Action Tasks assigned by Dispatch based on Action Events

Action Event	Police Unit Task	Fire Unit Task
Fire In Progress	Crowd Control	Fight Fire
Fire Out	Crowd Control	Clean Site
Emergency Over	Return To Base	Return To Base

Once it has assigned tasks to the units, the role of the Dispatcher is complete until the next report is received from the site of the emergency.

When the Dispatch model receives the Action Event from the Gateway it maps it into a Task for each of the units to perform. The type of Task depends on the Action Event and the unit type. The Dispatch model exports the Location ID of the Action Event to the PoliceUnit and FireUnit without changing its value.

The behaviour of the Dispatch model is not dependent on any initial state information. It does not request any data from the MessageHolder on the Gateway.

5.2.5 Responsibilities of Police and Fire Units

The Fire Unit entity is responsible for responding to instructions from the Dispatcher. The instructions from the Dispatcher often require the Fire Unit to move to a new location. The time at which the Fire Unit arrives at the new location depends on the location of the Fire Unit when it receives the new instruction from the Dispatcher. The Fire Unit therefore needs to be aware of its current location before it can respond to the new instruction.

The Police Unit entity is also responsible for responding to instructions from the Dispatcher. If the instructions require it to move to a new location, the amount of time it will take to arrive depends on the location of the Police Unit when it receives the new instruction.

The behaviour of the PoliceUnit and the FireUnit are similar. They require initial state data from the Gateway. Each unit requests its Unit data from the MessageHolder, passing in its unique Unit ID. The Unit Data received from the MessageHolder contains the current task being performed by the unit and the location of the unit. The units then request their current location coordinates from the Message Gateway.

When the Dispatch model sends the individual Tasks to the units along with the Location ID of the current Action Event, the units request the location coordinates of that location from the MessageHolder. They calculate the time required to get from their current location to the location of the Action Event and passivate for that length of time. When they activate again they output the task and the location to which they are heading. The output file contains the time at which the values were output as well as the output values. In this way the DEVS Model is able to calculate the time at which the unit will arrive at the new location and begin its new Task. This output information is then parsed by the DEVS Bridge and becomes input to the model during the next execution of the DEVS Model.

5.3 Results of DEVS Model Execution

The Emergency Services DEVS model was implemented and used with the DEVS Bridge architecture to execute the scenario. A number of steps must be executed to complete the scenario. The steps are

listed here, and expanded upon in the sections below. These steps assume that the script and resource set have been loaded into the DEVS Bridge.

1. The DEVS Bridge generates the values for the Report Message and State data.
2. The Report Message and State data are saved in the MA file defining the DEVS coupled model.
3. The DEVS Bridge generates the workspace definition (XML) and archive files.
4. The DEVS Bridge creates the workspace on the RISE server.
5. The DEVS Bridge posts the DEVS Model on the RISE server.
6. The DEVS Bridge executes the DEVS Model on the RISE server.
7. The DEVS Bridge gets the results of the execution from the RISE server.
8. The DEVS Bridge deletes the workspace so that it can be created for the next step.

The first step is to generate the input to the DEVS Model using the DEVS Bridge. A sample of the data is shown in the figure below. It is included into the MA file which defines the DEVS Coupled Model, replacing any values that were there previously.

```
[gway]
event : <FireInProgress,100>
units : <1000,Idle,200,200><1001,Idle,300,300>
locations : <100,43.987,73.234><200,23.152,55.934><300,98.234,42.654>
```

Figure 19 DEVS Model Input

The [gway] text indicates that the lines below it are parameters for the *gway* model component, which is an instance of the Gateway model. The *event* line contains the text for the Report Message. *FireInProgress* represents the Action Event category code value. 300 is the Location ID describing the location of the fire.

The *units* and *locations* lines contain the initial state information. For this model the initial state information describes all of the units in the simulation and all of the locations. These lines are formatted as described in section 4.2. The last location, 300, is the location of the fire described in the *event* line.

Once the DEVS Model input has replaced the contents of the DEVS coupled model found in the MA file, the DEVS Bridge prepares an archive file and an XML file to define the model on the server. This archive file contains the coupled model, any external event files, and all of the source code for the DEVS atomic models (.cpp and .h files). The XML file lists all of the files in the archive file. It also identifies the model

components and allocates them to the processors on the RISE server. A sample of the XML file is shown below.

```
<?xml version="1.0"?>
  <ConfigFramework>
    <Doc/>
    <Files>
        <File ftype="src">Dispatch.cpp</File>
        <File ftype="hdr" class="Dispatch">Dispatch.h</File>
        <File ftype="src">FireUnit.cpp</File>
        <File ftype="hdr" class="FireUnit">FireUnit.h</File>
        <File ftype="src">Gateway.cpp</File>
        <File ftype="ev">Gateway.ev</File>
        <File ftype="hdr" class="Gateway">Gateway.h</File>
        <File ftype="ma">Gateway.MA</File>
        <File ftype="sup">MessageBody.txt</File>
        <File ftype="sup">notes1.txt</File>
        <File ftype="src">PoliceUnit.cpp</File>
        <File ftype="hdr" class="PoliceUnit">PoliceUnit.h</File>
        <File ftype="sup">Target_Out.txt</File>
    </Files>
    <Options>
        <TimeOp>null</TimeOp>
        <ParsingOp>false</ParsingOp>
    </Options>
    <DCDpp>
        <Servers>
            <Server PORT="8080" IP="localhost">
                <MODEL>gway</MODEL>
                <MODEL>fire</MODEL>
                <MODEL>dis</MODEL>
                <MODEL>police</MODEL>
            </Server>
        </Servers>
  </DCDpp>
</ConfigFramework>
```

Figure 20 XML Workspace Creation File

The *ConfigFramework* element is the root element of the file. The *Doc* element contains documentation, or a text description that is displayed on the RISE server explaining the purpose of the simulation. The *Files* element marks the group of definition of individual files. Each file is described in a File element. The

ftype attribute describes the type of file and is based on the file extension. Possible values are shown in Table 5.

Table 5 Workspace File Types

File Type Description	File extension	Ftype value
C++ Header file	.h	hdr
C++ body file	.cpp	src
DEVS Model definition	.ma	ma
Event file	.ev	ev
Other file types, e.g. text file containing notes on the file.	.txt	sup

Elements which describe header files also have a Class attribute, which provides the name of the class.

The *Options* element contains options for the execution of the model. The *TimeOp* element defines the stop time for the scenario. The *ParsingOp* element indicates whether the model should print extra information when parsing occurs.

The *DCDpp* element is used to describe the distribution of the model components when parallel DEVS is used. In the case of the Emergency Service, all components are defined on the same server. The *Servers* element encompasses the list of available servers. Within the *Servers* element a *Server* element is created for each Server used in the simulation. The *Server* element identifies the machine where the component runs, specifying the Port and the IP address in the attributes on the element. Within the Server element, each of the *Model* elements defines a component that is defined in the MA file.

Once the XML and archive files have been created, the DEVS Bridge creates the workspace and posts it to the RISE Server. When this is done, the DEVS Bridge displays the message posted to the server, and displays the results. Below is a sample of the output from creating the workspace on the RISE Server.

```
New message type: PUT Label: Create Workspace sent to:
http://134.117.53.66:8080/cdpp/sim/workspaces/elizabeths/dcdpp/Dispatch_Model/<Conf
igFramework><Doc></Doc><Files><File ftype="src">Dispatch.cpp</File><File
ftype="hdr" class="Dispatch">Dispatch.h</File><File
ftype="src">FireUnit.cpp</File><File ftype="hdr"
class="FireUnit">FireUnit.h</File><File ftype="src">Gateway.cpp</File><File
ftype="ev">Gateway.ev</File><File ftype="hdr" class="Gateway">Gateway.h</File><File
ftype="ma">Gateway.MA</File><File ftype="sup">MessageBody.txt</File><File
ftype="sup">notes1.txt</File><File ftype="src">PoliceUnit.cpp</File><File
ftype="hdr" class="PoliceUnit">PoliceUnit.h</File><File
ftype="sup">Target_Out.txt</File></Files><Options><TimeOp>null</TimeOp><ParsingOp>f
alse</ParsingOp></Options><DCDpp><Servers><Server IP="localhost"
PORT="8080"><MODEL>gway</MODEL><MODEL>fire</MODEL><MODEL>dis</MODEL><MODEL>police</
MODEL></Server></Servers></DCDpp></ConfigFramework>
```

Figure 21 DEVS Bridge Output from creating Workspace on the RISE server.

The first line of text describes the RESTful message type, which in this case is a “PUT” model. The “Label” is a descriptive label used for annotating the message for the user’s reference. The “sent to” text prefaces the HTTP address of the server where the message is sent. This is followed by the body of the message, which is the XML from Figure 20 XML Workspace Creation File. The “Response to message” text indicates the response received from the RISE server after it received the PUT message. “Create Workspace” is a repetition of the Label. The line “No response received” indicates that no message was received. This is expected for a PUT message. If an error occurred the text of the error message would be displayed.

Once the model has been posted and executed, the DEVS Bridge is used to get the results from the server. The results include log files that report on the parsing and execution of the model, and the out file generated by the DEVS Model. The out file contains the values that appear on all output ports defined on the top level model in the DEVS Model. For the Emergency Services model, output ports are defined for each of the Police and Fire Units in the model. Each unit has two output ports. The loc_out, or location output port, produces the object id of the location of the unit. The task_out, or task output port, produces an integer which maps to the Action Task Category Code, which represents the task being performed by the unit. All outputs on the ports are printed to the out file from the DEVS model. The output includes the time at which the output is produced.

A sample of the output file from the DEVS Model is shown in the figure below.

```
00:00:00:005 police_loc_out 0
00:00:00:005 police_task_out 0
00:00:00:006 fire_loc_out 0
00:00:00:006 fire_task_out 0
00:07:17:000 fire_loc_out 300
00:07:17:000 fire_task_out 4
00:35:17:000 police_loc_out 300
00:35:17:000 police_task_out 3
```

Figure 22 Sample DEVS Model Output

The first four messages are the result of the way initialization is implemented. They are produced by the internal transitions by the Police and Fire Units when they load their initial state information, including their current location and task. These operations are scheduled a few milliseconds into the scenario to ensure that the Gateway model has initialized and parsed the input Report Message and state information.

The last four messages are the ones that are parsed by the DEVS Bridge. They represent the time at which the units would arrive at their new location and being to perform their new task. This output was produced in response to the initial Report Message of the Fire in Progress. Both the Police Unit and Fire Unit report their new location as 300, which is the location of the fire. Their different tasks, 4, and 3, correspond to FightFire and Crowd Control respectively. The different times at which they arrive at the scene reflect the difference in the original locations of the units and the different speeds at which they travel.

The DEVS Bridge parses this output. It maps the task_out values to the Action Task Category Code. This information, the time, location id and the task code, is saved in the data model maintained by the DEVS Bridge. The location ids and Action Task Category Codes are used to generate the input to the simulation the next time a Report Message is processed by the DEVS Model. In a simulation where there is an external simulation, instead of a scripted scenario run by the DEVS Bridge itself, this data would then be output to the external simulation.

5.4 Executed Scenarios

To validate the architecture, a number of scenarios were executed. In addition to the basic scenario described above, two additional scenarios were executed. In the second scenario a second Fire Unit was

requested to report to the scene to fight the fire. In the third scenario a second Fire Unit was requested, and an ambulance was called to attend to casualties at the scene and transport them to a hospital.

The set of messages used in the first scenario is shown in Table 6 Scenario 1 Messages. Each message contains the Action Event Category Code value in the message and the unique id of the location at which the Action Event takes place. The list of Locations is given in Table 10 Locations for Resource Set.

Table 6 Scenario 1 Messages

Time	Message and Location
00:01:00	FireInProgress at: 100
00:30:00	FireOut at: 100
01:00:00	EmergencyOver at: 100

In order to add a second Fire Unit to the scenario a number of modifications were made to the DEVS model described above. New ports were added to the Dispatch model to communicate with the second Fire Unit. The MA file was updated to define a second instance of the Fire Unit atomic model and connect it to the new ports on the Dispatch model. Additional output ports were defined on the Top Coupled Model so that outputs from the second Fire Unit would be added to the out file. An additional Action Event Category Code was added, RequestSupportFire. This code represents a request from the Fire Unit on the site of the emergency for additional support.

The set of messages in the second scenario are shown in Table 7 Scenario 2 Messages.

Table 7 Scenario 2 Messages

Time	Message and Location
00:01:00	FireInProgress at: 100
00:15:00	RequestSupportFire at: 100
00:50:00	FireOut at: 100
01:00:00	EmergencyOver at: 100

The third scenario added an Ambulance Unit to the DEVS model. The Ambulance Unit is called to the scene of an emergency to treat injured parties at the site. In the case where an injury is serious, the

Ambulance may be ordered to take the patient to a hospital. In order to accommodate the behaviour of this unit changes were made to the rest of the model. Two new Action Event Category Codes were added: RequestSupportMedivac, which is the call for an Ambulance, and MedivacRequest, which indicates that the Ambulance is en-route to the hospital. Two new Action Task Category Codes were added to represent tasks assigned to the Ambulance by the Dispatcher, TreatOnSite and Medivac. The behaviour of the Dispatcher was modified to issue these new tasks. The new behaviour of the Dispatcher is shown in the table below.

Table 8 Dispatcher Tasking of Ambulance Unit

Action Event	Ambulance Unit Task
RequestSupportMedivac	TreatOnSite
Medivac	Medivac
Emergency Over	Return To Base

The Dispatcher does not assign any tasks to the Ambulance Unit for the Fire In Progress or Fire Out Action Events. New ports were added to the Dispatcher to send data to the Ambulance Unit. The MA file was modified to create the Ambulance Unit instance and new ports were added to the Top model so that the output from the Ambulance unit would appear in the out file.

The messages in the third scenario are shown in the table below.

Table 9 Scenario 3 Messages

Time	Message and Location
00:01:00	FireInProgress at: 100
00:15:00	RequestSupportMedivac at: 100
00:20:00	RequestSupportFire at: 100
00:30:00	Medivac at: 500
01:00:00	FireOut at: 100
01:30:00	EmergencyOver at: 100

The three scenarios were executed using the same basic set of resources. The Resource Set contained a set of locations and units. However, in order to collect more data about the execution of the scenario

three versions of the Resource Set were created. Each copy of the Resource Set had the same set of locations and units, but different coordinates were assigned to each of the locations. The coordinates of each Location were assigned by representing the city as a grid. The latitude and longitude values in the grid range from 0 – 100.

In the first Resource Set, the locations were scattered randomly across the grid. In the second Resource Set, the site of the fire, the Zellers location, was placed in one corner of the grid. The Fire and Police Stations and the hospital were lined up diagonally across the grid. In the third set the Zellers location was set in the middle of one edge of the grid. The other locations were spread along the opposite edge of the grid. The complete set of locations and the three sets of coordinates are listed in the table below.

Table 10 Locations for Resource Sets

Unique Id	Location Name	Resource Set 1 Coordinates	Resource Set 2 Coordinates	Resource Set 3 Coordinates
100	Zellers	(44, 73)	(5, 98)	(50, 99)
200	Police Station 1	(23, 56)	(60, 80)	(10, 10)
300	Fire Station 1	(98, 43)	(20, 25)	(35, 10)
400	Fire Station 2	(66, 10)	(92, 10)	(70, 10)
500	Community Hospital	(85, 35)	(45, 7)	(95, 10)

The Resource Sets used to execute the scenarios contained the same set of four units. The home locations of the units are included in the definition of the unit. When the ReturnToBase Action Task is assigned to a unit it calculates the time to move from its current location to its home location. The set of units is listed in the table below.

Table 11 List of Units

Unit Name	Unit Type	Unique ID	Home Location	Home Location ID
Car1	Police	1000	Police Station 1	200
Truck1	Fire	1001	Fire Station 1	300
Truck2	Fire	1002	Fire Station 2	400
Ambulance1	Ambulance	1003	Fire Station 2	400

The Ambulance is located at the same Fire Station as the second Fire Unit, Truck2.

When calculating the time required to perform an action, the distance between the locations is one of the factors. The other factor is the speed at which the unit moves. The speed is a value defined in the MA file.

5.5 Results and Analysis

The results of executing the messages are shown in the table below. The times listed correspond to Resource Set 1, Resource Set 2 and Resource Set 3.

Table 12 Scenario 1 Timeline

RS1	RS2	RS3	Event	Action Task
00:01:00	00:01:00	00:01:00	FireInProgress report sent to Dispatch	
00:02:00	00:04:00	00:07:00	Car1 arrives at Zellers	Crowd Control
00:06:00	00:07:00	00:08:00	Truck1 arrives at Zellers	FightFire
00:30:00	00:30:00	00:30:00	FireOut report sent to Dispatch	
00:30:00	00:30:00	00:30:00	Truck1 changes task	Clean Site
01:00:00	01:00:00	01:00:00	EmergencyOver report sent to Dispatch	
01:01:00	01:03:00	01:06:00	Car1 arrives at Police Station 1	Return To Base
01:05:00	01:06:00	01:07:00	Truck1 arrives at Fire Station 1	Return To Base

The results from running the second scenario are shown in the table below.

Table 13 Scenario 2 Timeline

RS1	RS2	RS3	Event	Action Task
00:01:00	00:01:00	00:01:00	FireInProgress report sent to Dispatch	
00:02:00	00:04:00	00:06:00	Car1 arrives at Zellers	Crowd Control
00:06:00	00:07:00	00:07:00	Truck1 arrives at Zellers	FightFire
00:15:00	00:15:00	00:15:00	RequestSupportFire report sent to Dispatch	
00:21:00	00:27:00	00:22:00	Truck2 arrives at Zellers	Fight Fire
00:50:00	00:50:00	00:50:00	FireOut report sent to Dispatch	

00:50:00	n/a	n/a	Truck1 changes task	Clean Site
00:50:00	n/a	n/a	Truck2 changes task	Clean Site
01:00:00	01:00:00	01:00:00	EmergencyOver report sent to Dispatch	
01:01:00	01:03:00	01:06:00	Car1 arrives at Police Station 1	Return To Base
01:05:00	01:06:00	01:07:00	Truck1 arrives at Fire Station 1	Return To Base
01:06:00	01:12:00	01:07:00	Truck2 arrives at Fire Station 2	Return To Base

The results of executing the third scenario are shown in the figure below.

Table 14 Scenario 3 Time Line

RS1	RS2	RS3	Event	Action Task
00:01:00	00:01:00	00:01:00	FireInProgress report sent to Dispatch	
00:02:00	00:04:00	00:06:00	Car1 arrives at Zellers	Crowd Control
00:06:00	00:07:00	00:07:00	Truck1 arrives at Zellers	FightFire
00:15:00	00:15:00	00:15:00	RequestSupportMedivac report sent to Dispatch	
00:19:00	00:27:00	00:20:00	Ambulance1 arrives at Zellers	TreatInPlace
00:20:00	00:20:00	00:20:00	RequestSupportFire report sent to Dispatch	
00:27:00	00:32:00	00:27:00	Truck2 arrives at Zellers	Fight Fire
00:30:00	00:30:00	00:30:00	Medivac report sent to Dispatch	
00:33:00	00:36:00	00:36:00	Ambulance1 arrives at Community Hospital	Medivac
01:00:00	01:00:00	01:00:00	FireOut report sent to Dispatch	
01:00:00	n/a	n/a	Truck1 changes task	Clean Site
01:00:00	n/a	n/a	Truck2 changes task	Clean Site
01:30:00	01:30:00	01:30:00	EmergencyOver report sent to Dispatch	
01:31:00	01:33:00	01:36:00	Car1 arrives at Police Station 1	Return To Base
01:35:00	01:36:00	01:37:00	Truck1 arrives at Fire Station 1	Return To Base
01:36:00	01:42:00	01:37:00	Truck2 arrives at Fire Station 2	Return To Base
01:34:00	01:32:00	01:31:00	Ambulance1 arrives at Fire Station 2	ReturnToBase

In each of the three scenarios, the runs with the three different sets of locations resulted in different times of arrival of the units vary due to the differing distances between the locations. The use of the different Resource Sets demonstrates how a single model can be run using different scenario definition files, such as MSDL, to reflect the differences in scenarios. The use of the existing model in different scenarios using different sequence of messages demonstrates how a model can participate in a variety of scenarios with different series of input events from the rest of the synthetic environment.

The case study selected to validate the architecture involves civilian entities responding to an emergency. This was done because of a perceived need to integrate the C2 systems of civilian agencies with military agencies. However, as noted there are problems using C-BML with civilian agencies. This resulted in the need to use custom category codes. If the case study had involved a military unit on patrol responding to a military emergency, the C-BML category codes could have been used in the scenario to validate the architecture. Alternately, if the external simulations used a custom schema developed by extending C-BML with custom category codes, no special handling would be required for the simulation.

5.6 DEVS Bridge Input Discussion

As described in section 3.2.1, it was decided that rather than having an external simulation drive the DEVS Bridge, the DEVS Bridge would be driven by a script. To accommodate this, the ability to create a script, save it to a file, and load it from a file, was built into the DEVS Bridge. The script can be viewed either via a dialog listing the Report Messages in the script, or displayed on the DEVS Bridge interface. A sample listing of a script is shown in the figure below.

```
Messages:
   00:01:00 - FireInProgress at: 300
   00:30:00 - FireOut at: 300
```

Figure 23 Script Listing

Each line under the Messages label describes one Report Message. The time is the time at which the Report should be sent to the DEVS Model. This value is mostly used to give an order to the scenario. If the execution of the script was automatic, this value would control the time at which the Report Message is executed by the DEVS Model. The text following the dash (“-“) is the Action Event Category Code for the Report Message. The number that follows the “at:” text is the location id for the location

where the Action Event is occurring. For the “EmergencyOver” category code, the location id is the location where the original Action Event occurred. The DEVS Model calculates the time required for the units to return to their original locations.

The scenario is executed by creating input to the DEVS Model using each of the Report Messages. The execution of a Report Message is controlled by manually generating output on the DEVS Bridge. Using the script allowed the execution of the scenario to be controlled rather than having to respond to messages being received from an external process. Having the control of the script execution being controlled by the user was also necessary because the process of saving the Report Message as part of the DEVS Model and posting the DEVS Model to the RISE server was also a manual process.

The scripting capability replaced the reception of messages from an external simulation. Without the internal script capability, the DEVS Bridge would operate by receiving messages from an external simulation. It would define a Web Endpoint that could be invoked by other services. The endpoint would provide methods for delivering XML-formatted C-BML messages.

Once the DEVS Bridge received the C-BML message it would have to parse it before the message could be passed to the DEVS Model. This would require implementing a full C-BML message parser that could extract all of the required information from the passed-in message. As discussed in section 3.4, the full C-BML schema is extensive and complex to parse.

Once the C-BML message had been parsed and send to the DEVS Model for execution, the results would be parsed into the internal data model used by the DEVS Bridge. The DEVS Bridge would then have to convert the output from the DEVS Model into a C-BML message. This message would then be either returned directly to the external simulation which sent in the Report, or published in a manner that it would be available to all simulations in the synthetic environment.

One possible configuration for allowing simulations to communicate with each other in a larger environment is shown in the figure below.

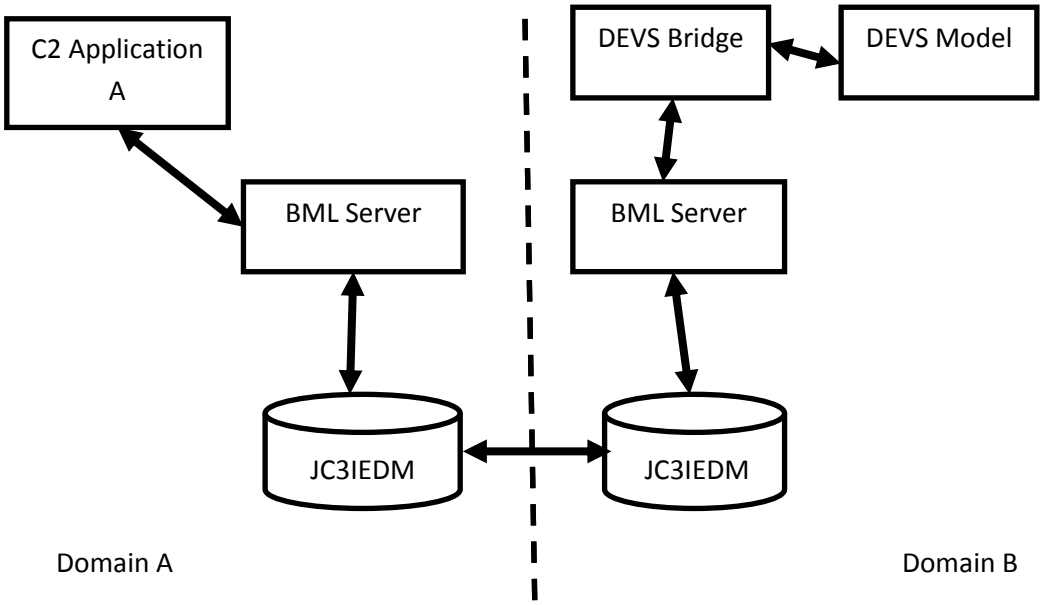

Figure 24 DEVS Bridge communicating via BML Server and Database Replication

In this version of the configuration, the entities in Domain B are modelled using a DEVS Model. The Applications use the BML Server to publish Orders, Reports and Requests to the database, which is defined based on the JC3IEDM model. Data published to a database in one domain is replicated to a database in another domain according to the rules created when the connection between the databases is created. For example, the database in Domain A may send Orders to the database in Domain B, while the database in Domain B sends Reports back to Domain A.

The DEVS Bridge and DEVS Models run as the applications in Domain B to simulate a unit receiving Orders via a C2 system and posting the unit's response to the Orders. The DEVS Bridge subscribes to notifications from the BML Server. When the C2 Application in Domain A publishes an Order that affects the entities modelled by the DEVS Model, the Order is sent to the BML Server, which writes it to the database. The Order is replicated from the database in Domain A to the database in Domain B. When the Order arrives in Domain B the BML Server receives notification of the new Order and informs its subscribers. The DEVS Bridge receives the notification of the new Order and retrieves the new Order from the database in C-BML format. It parses the order, extracting the information required by the DEVS Model. Once the DEVS Model has calculated the response of the entities it models, the DEVS Bridge converts the output from the DEVS Model into C-BML messages. It then publishes the C-BML messages

to the BML Server, which writes the data to the database. Database replication publishes the information to the rest of the databases in the environment.

Another possibility for replacing the scripting interface is to implement a SOAP interface to receive messages from other simulations. Since SOAP is an XML format it can accept messages formatted using C-BML to define the schema.

The DEVS Bridge Architecture could be used to define a C-BML interface for other simulations besides DEVS simulations. Further investigation would be required to determine what constraints would be required on the other models in order to be used with the DEVS Bridge architecture.

5.7 Traceability using the DEVS Bridge

After the execution of a simulation, it is beneficial to be able to review the progress of the simulation and the changes of the states of the modelled entities throughout the simulation. The DEVS Bridge as implemented has a text pane where the messages being sent and the model values are displayed. The text in this pane can be saved to a file. Using this output the changes of the state of the entities and their attributes can be traced during execution of the scenario or after execution of the scenario is complete.

The DEVS Bridge can save the resource set and scripts as XML Files. Further work on the DEVS Bridge could automate saving the messages and state data to file as they are processed during the scenario to provide traceability.

5.8 Discussion

Most of the JC3IEDM codes have a Not Otherwise Specified (NOS) value. In the case where civilian and military entities must coordinate, the messaging schema to be used could be extended by defining sub-types of Actions and Action Tasks. The attributes in the regular elements that use the category codes would be set to NOS. The type extensions would use values from custom category codes defined for the exercise. The military C2 systems would ignore the extensions in the messages. The C2 user interfaces would display the Fire Fighter units as NOS units. This would allow the military commanders to view the location of the Fire Fighter units, although without details of the unit types. Other civilian C2 systems would be able to parse the extended data and would display the detailed unit types.

The DEVS Bridge architecture could be used as an external-facing, web-enabled interface for other simulation systems. It could be adapted for other message-based protocols. The architecture, as it is

proposed, is limited by the requirements on the DEVS Models: the need to load all state data at the start of the message execution and the fact that processing is limited to responding to one input message from the external environment. For a complex DEVS model with a large number of atomic models, the overhead of importing and exporting the state data and replicating it on the DEVS Bridge may be impractical.

6 Conclusions

6.1 Summary of Motivation

Simulation and Modeling are increasingly being used to study complex scenarios for the purposes of planning and training. The use of computer simulation allows the evaluation of a process that in real life consumes expensive or dangerous resources. New systems or processes can be modelled prior to implementing or building them so that designers can evaluate how well they function, potential bottlenecks, or performance of the system under stress. This allows the designers to improve the system or process before it is put into place.

Simulation is being used for training purposes, again in situations where execution of the real scenario would require the consumption of rare or expensive resources, such as surgical trainers or veterinarian simulators. Simulation is also used for training of personnel in the handling of equipment in dangerous situations, such as firefighting. Simulation is used by military organisations for planning and mission rehearsal to evaluate possible courses of actions and their results. It is also used by civilian agencies for planning responses to emergency situations or natural disasters such as fire or flood.

Simulations have been built using a variety of technologies, many of them using proprietary data formats and messaging. However, being able to combine simulations allows the creation of larger synthetic environments that can be used to study more complex problems. This has led to the development of standards that allow simulations to interact. These standards, such as HLA/DIS, specify how the simulations exchange data. With the growth of Web technologies the Coalition Battle Management Language (C-BML) specification is evolving as an XML-based messaging format that can be exchanged between real and simulated systems. The Military Scenario Definition Language (MSDL) is a complementary specification to C-BML that is used to initialize the entities in a simulation. The use of or compliance to some form of interoperability specification is important for emerging simulations so that they may be used as part of a larger synthetic environment.

Discrete Event Simulation (DEVS) is a modelling formalism that can be used to model a wide range of systems. DEVS models can be used to study performance loads on a system with shared resources. Cellular DEVS can be used to model systems that can be represented as discrete units, such as biological systems. This allows modeling of the growth pattern of cancer cells. It can also be used to model traffic patterns and evacuation of buildings or airplanes.

In order to make DEVS technologies available to a wide range of clients on different hardware and software platforms the RESTful Interoperability Simulation Environment (RISE) was developed. It hosts DEVS models and provides a web-enabled interface for accessing the models and executing them. Adding support for other types of interoperability would increase the range of scenarios where DEVS Models could be integrated with other systems and or simulations. This thesis work was done in order to provide extended interoperability.

This thesis defines an architecture that adds a service to act as an interface to the RISE server. The service insulates the model on the RISE server from the specifics of the synthetic environment. The service communicates with the synthetic environment using the required interoperability protocol, and translates it to the format required to communicate with the DEVS model.

Many simulations have been built for military scenarios, so many interoperability standards are based on the requirements for modeling military systems (HLA/DIS) or communicating between military systems (C-BML, MSDL.) While some simulations have been built to model civilian scenarios and civilian organisations, there has not been a lot of modeling of civilian entities interacting with the military systems using standards. For this reason a civilian emergency use case was selected for modeling.

The C-BML specification and the use of MDSL files allow simulations to work with real-world systems. Therefore the use of C-BML and MSDL were studied for their applicability for use with DEVS models on the RISE server. They were also studied for their use in reporting by and tasking of civilian entities.

6.2 Summary of Goals

6.2.1 Goals

The main goal of the thesis was to define an architecture that added a simulation interoperability interface to DEVS Models running on the RISE server. This included studying the requirements of building this architecture around the RISE server. In order to validate the architecture a case study was defined and executed. Once the basic scenario was defined extended versions of the case study were created to verify that changing the scenario did not require modifications to the basic architecture.

Another goal was to study the MSDL and C-BML specifications for use with civilian simulations. These specifications were defined for military simulations. However, emergency scenarios often require military and civilian agencies to work together.

A related goal was to study the MSDL and C-BML standards for their usability in initializing and coordinating with DEVS models running on the RISE server.

6.2.2 How Goals Were Achieved

The RISE server provides an environment where DEVS models can execute. In order to be able to interact with other simulations, the architecture requires a component which provides the ability to communicate with them. The capabilities of this component were defined by identifying the work that this this component would need perform in order to support the execution of a scenario. The behaviour of this component was modeled using UML diagrams. These diagrams then drove the design.

The category codes for entities in the C-BML and MSDL standards are taken from the JC3IEDM specification. These category codes were examined for values that could be used to define civilian emergency response agencies. The category codes were also examined for values that could be used to assign tasks to these agencies. Some shortfalls in the category codes were identified. The results are presented in section 3.6.4.

The key entities in the C-BML and MSDL specifications were discussed in section 3.4.1. The intent was to use them to validate the proposed architecture. However, given the complexity of the specifications the decision was made not to use the full specifications. Instead simplified schemas that modeled the key concepts in the standards were defined for use in validating the architecture.

A basic emergency scenario involving civilian emergency response organisations was defined and used to validate the architecture. The scenario was extended with additional messages and entity types. These changes were made to verify that the architecture could be used unchanged with different models and scenarios.

6.3 Summary of Contributions

6.3.1 Topics Investigated

This thesis proposes an architecture for adding simulation interoperability to the RISE server. The architecture proposes creating a bridge component to act as an interface to the RISE server from

external systems. The bridge component separates the logic for communicating with external systems from the logic to communicate with the RISE server. The architecture was validated by implementing a component that was designed to communicate with an external system using structured messages, such as messages defined using the C-BML specification. The bridge component was used to validate the architecture by creating a model of a civilian emergency scenario. The scenario was then extended by adding new entities and events. The scenarios were added and executed without requiring changes to the bridge component, thus validating the proposed architecture.

The C-BML and MSDL specifications were examined for use in a scenario involving civilian emergency service agencies. The category codes defined by JC3IEDM that would be used in defining and tasking the emergency service units were examined. A number of areas were identified where the category codes were lacking values that would be required for civilian agencies. These concerns were discussed in section 3.4.

The use of MSDL and C-BML in initializing and communicating with the DEVS model were investigated. While the specifications were found to be more complex than was required, they served as the basis for defining simplified schemas that were used during the validation of the architecture. The DEVS Model was executed multiple times, using the scenario definition schema to supply the initial state information for the DEVS Models. The input to the DEVS Model was supplied using messages defined according to the custom standard. The use of these custom schemas demonstrated that the execution of DEVS Models on the RISE server is well suited to the use of scenario definition files and structured input messages.

6.4 Future Research

6.4.1 Proposed Areas of Investigation

The proposed architecture was developed to provide an interoperability interface to the DEVS models on the RISE server. The architecture was developed by considering the requirements to provide a Web-enabled interface exchanging XML-based messages. The implemented component acted as a bridge between the DEVS model and the external environment. One avenue for further investigation would be to investigate how the architecture could be used to implement an interoperability interface using other technologies. Instead of serving as a simple bridge, the new component could be an HLA or DIS-compliant federate. The requirements on the behaviour of this component would likely be different than the requirements which were identified for the message-based bridge component.

The bridge component that was developed as part of this thesis used the execution of a pre-defined script to simulate input from the external synthetic environment. The bridge component could be modified to parse messages from a schema defined using the C-BML specification.

As discussed in section 2.2.1 the Center of Excellence at George Mason University has implemented a C-BML Server. Clients of the server subscribe to notification of incoming messages. As an alternative to receiving and processing XML-based C-BML messages, the bridge component could be modified to subscribe to the C-BML server and publish data to the database.

6.4.2 Potential Benefits

By defining an architecture that adds a simulation interoperability interface to the RISE server, this thesis provides a mechanism to incorporate DEVS models into a wider range of possible simulation scenarios. Cell-DEVS models of natural disasters such as forest fires could be incorporated into a synthetic environment used for planning emergency responses. Models of how flood waters rise or forest fires spread could be used to calculate the progress of the disaster. This information could be provided to other simulations such as Agent Based simulations that model the behaviour of people in the area, or supply information to a Computer Generated Force (CGF) system or Command and Control (C2) system that affects the orders given to simulated emergency responder agencies. The architecture could also be used to bridge between two different DEVS models simulating different parts of the synthetic environment. Each DEVS model would supply its output to an external coordinating system which would then calculate messages to be sent to other simulations.

7 References

[1] C. Blais et al, “An Architecture for Demonstrating the Interplay of Emerging SISO Standards,” in *Fall Simulation Interoperability Workshop* (*FallSIW*), Orlando, FL, 2006. Available: http://www.sisostds.org/DigitalLibrary.aspx?EntryId=27053 file 06F-SIW-069 Interplay.pdf.

[2] G. Wainer, *Discrete-Event Modeling and Simulation: A Practitioner’s Approach*, CRC Press, 2009.

[3] *OneSAF I2S Development With Eclipse*, [Online]. September 2012, Available: http://www.onesaf.net/community/static/web/Amphitheatre/202_OneSAF_TEM_2012_I2S_Eclipse.pdf

[4] VR-Forces Versions [Web Page]. Available : http://www.mak.com/support/product-versions/simulate/vr-forces.html

[5] D. MacQuarrie et al, “Simulation-C2 Interoperability Through Data Mediation: the Virtual Command and Control Interface,” in *Proceedings of the 2008 Summer Computer Simulation Conference,* Edinburgh, Scotland, Society for Modeling & Simulation International, ©2008.

[6] *Simulation Interoperability Standards Organization (SISO) Guide for Base Object Model (BOM) Use and Implementation,* SISO-STD-003.1-2006,31 March 2006

[7] “Hypertext Transfer Protocol,” *Wikipedia* Retrieved 2014, February 24. Retrieved from http://en.wikipedia.org/wiki/Http.

[8] *Simulation Interoperability Standards Organisation (SISO) Standard for Military Scenario Definition Language (MSDL)*, SISO-STD-007-2008, 14 October 2008.

[9] G. Wainer, “Modeling and Simulation of Complex Systems with Cell-DEVS”, *Proceedings of the 2004 Winter Simulation Conference*, 5-8 Dec, 2004

[10] Al-Zoubi, K et al, “RISE: REST-ing Heterogeneous Simulations Interoperability”, in *Proceedings of the 2010 Winter Simulation Conference,* Baltimore, Maryland, USA (2010).

[11] C. Timmons, “Web Service-Based Distributed Simulation of Discrete Event Models,” M.S. thesis, SYSC, Carleton University, Ottawa, ON, 2013.

[12] *Guidance, Rationale, and Interoperability Modalities for the Real-time Platform Reference Federation Object Model (RPR FOM)*, Version 1.0, 10 September 1999.

[13] *Multilateral Interoperability Programme (MIP) Implementation Guidance,* MIG; Edition 3.1.10*,* 25 April 2013

[14] *The Joint C3 Information Exchange Data Model,* JC3IEDM Main, 14 February 2012.

[15] *Simulation Interoperability Standards Organization (SISO) Guide for: Coalition Battle Management Language (C-BML) Phase 1,* SISO-GUIDE-002-2012-DRAFT, 4 April 2012.

[16] C. Blais et al, *"*Coalition Battle Management Language (C-BML) Phase I Standard: Trial Use Findings and Next Steps," *NPS Papers and Presentations for Simulation Interoperability Standards Organization's Simulation Interoperability Workshop,* [Online] 2011, Available: http://hdl.handle.net/10945/30788

[17] B. Gautreau et al, "Lessons learned from NMSG-085 CIG Land Operation demonstration," in *Spring Simulation Interoperability Workshop 2013 (2013 Spring SIW)",* San Diego, CA., April 8-12 2013, pp. 176-185.

[18] *Department Of Defence Interface Standard: Common Warfighting Symbology (30 Jan 1999)*, MIL-STD-2525B, Jan 1999.

[19] J.M. Pullen et al, "Maturing Supporting Software for C2-Simulation Interoperation," in *2011 15th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2011,* Salford, United Kingdom, September 4-7 2011, pp. 148-154,

[20] M. Pullen et al, *Scripted Battle Management Language Web Service Version 1.0 Operation and Mapping Description Language*, IEEE Spring Simulation Interoperability Workshop, San Diego, March 2009.

[21] D. Corner, et al, "*Adding Publish/Subscribe to the Scripted Battle Management Language Web Service,* IEEE Spring 2010 Simulation Interoperability Workshop, Orlando, FL, 2010.

[22] J. Pullen et al, *Performance and Usability Enhancements to the Scripted BML Server*, IEEE Fall 2010 Simulation Interoperability Workshop, Orlando, FL, 2010

[23] M. Pullen and L. Nicklas, Maturing Supporting Software for C2-Simulation Interoperation Pullen, M and Nicklas, in *2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications (DS-RT),* Salford, Sept 4-7 2011, pp.148-154.

[24] Standardization for C2-Simulation Interoperation (MSG-085) Web Site, http://www.cso.nato.int/ACTIVITY_META.asp?ACT=1957, retrieved September 2013.

[25] F. Bowers et al, "JTLS-JCATS federation support of emergency response training," in *Proceedings of the 2003 Winter Simulation Conference,* New Orleans, Louisiana, 7-10 Dec. 2003, pp.1052-1060.

[26] A. Cowdale et al, "Simulation modelling in support of emergency fire-fighting in Norfolk," in *Proceedings of the 2003 Winter Simulation Conference, 2003,* New Orleans, Louisiana, 7-10 Dec. 2003, pp.1707-1710.

[27]T. Brady et al, "Emergency management: capability analysis of critical incident response," in *Proceedings of the 2003 Winter Simulation Conference, 2003,* New Orleans, Louisiana, 7-10 Dec. 2003 pp.1863-1867

[28] C. Tao et al, "Integration of GIS and Computational Models for Emergency Management," in *2008 International Conference on Intelligent Computation Technology and Automation (ICICTA),* Changsha, Hunan, China, 20-22 Oct. 2008, pp.255-258.

[29] Y. Huang et al, "Simulation and Evaluation for the Emergency Management under the Situation of Fatal Disaster," in *2010 International Conference on Multimedia Communications (Mediacom),* 7-8 Aug. 2010, pp.258-262.

[30] R. Aringhieri et al, "An integrated DE and AB simulation model for EMS management," in *2010 IEEE Workshop on Health Care Management (WHCM),* Venice, Italy, 18-20 Feb. 2010, pp.1-6.

[31] M. Rahmat et al, "Agent-based modelling and simulation of emergency department re-triage," in *Business Engineering and Industrial Applications Colloquium (BEIAC), 2013 IEEE* , 7-9 April 2013, pp.219-224.

[32] S. Hessam et al, “The Role of Collaborative DEVS Modeler in Federation Development,” in *Proceedings of the 1999 Fall Simulation Interoperability Workshop*, Orlando, FL,

[33] M. Saurabh et al, “DEVS-Based Simulation Web Services for Net-Centric T&E” in *Summer Computer Simulation Conference 2007 (SCSC 2007)* San Diego, CA, 2007.

[34] M. Tavanpour et al, “Simulation of Mobile Networks using Discrete Event System Specification Theory,” in *Proceedings of the 16th Communications & Networking Symposium*, ACM, (2013), page 13.

[35] K. Al-Zoubi et al, "Using REST Web-Services Architecture for Distributed Simulation," in *ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation, 2009. PADS '09*, 22-25 June 2009, pp.114-121

[36] J. Ribault et al, "Simulation Processes in the Cloud for Emergency Planning," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid),* 13-16 May 2012, pp.886-891.

[37] B. Zeigler et al, "The DEVS/HLA Distributed Simulation Environment and Its Support for Predictive Filtering", A Deliverable in Fulfillment of Advance Simulation Technology Thrust (ASTT) DARPA Contract K-N6133997K-0007, September 1998. Available: UnivArizonaCDRL2.pdf

[38] C. Pham et al, "HLA Support in a Discrete Event Simulation Lanugage," *3rd IEEE International Workshop on Distributed Interactive Simulation and Real-Time Applications, 1999, Proceedings.* p. 93-100. Greenbelt, MD, 1999.

[39] G. Zacharewicz et al, "Services Modeling and Distributed Simulation DEVS/HLA Supported," in *Proceedings of the 2009 Winter Simulation Conference*, 2009. Austin, Texas p. 3023-3035

[40] G. Zacharewicz et al, "GDEVS/HLA Environment: A Time Management Improvement", *Proceedings of the 17th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation*, 2005,

[41] G. Zacharewicz et al, "Flattening G-DEVS / HLA structure for Distributed Simulation of Workflows", *Journal for the Society for Computer Simulation International* Volume 84 May 2008, p. 197-213.

[42] G. Zacharewicz et al, "Lookahead Computation in G-DEVS/HLA Environment", *Simulation News Eurpoe Journal (SNE) 16,* 2 (2006) p. 15-24.

[43] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph D. dissertation, University of California, Irvine, CA. 2000

[44] P. Cappelaere et al, "Flow-enablement of the NASA SensorWeb using RESTful (and secure) workflows," in *Proceedings of the 2009 IEEE Aerospace conference*. Big Sky, Montana, 7-14 March 2009, pp. 1-7.

[45] M. Rouached et al, "RESTful Sensor Web Enablement Services for Wireless Sensor Networks," in *2012 IEEE Eighth World Congress on Services (SERVICES),* 24-29 June 2012, pp.65-72.

[46] Q. Zhan et al, "The research and implementation of a RESTful map mashup service," in *2010 Second International Conference on Communication Systems, Networks and Applications (ICCSNA),* June 29 2010-July 1 2010, pp.401-403.

[47] M. Lanthaler et al, "Aligning Web Services with the Semantic Web to Create a Global Read-Write Graph of Data," in *2011 Ninth IEEE European Conference on Web Services (ECOWS),* 14-16 Sept. 2011, pp.15-22.

[48] M. Tosic et al, "A RESTful technique for collaborative learning content transclusion by Wiki-style mashups," in *2011 5th IEEE International Conference on e-Learning in Industrial Electronics (ICELIE)* 7-10 Nov. 2011, pp.38-43.

[49] P. Bing , "An aggregation search engine based On RESTful Web services and Mashup," in *2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE),* 10-12 June 2011, pp.142-146.

[50] C. Bo et al, "RESTful Web Service Mashup Based Coal Mine Safety Monitoring and Control Automation with Wireless Sensor Network," in *2012 IEEE 19th International Conference on Web Services (ICWS),* 24-29 June 2012, pp.620-622.

[51] A. M. Winn, “Multimedia workshop: exploring the benefits of a visual scripting language”, *1998 IEEE Symposium on Visual Languages*, Halifax, NS, 01-04 Sep 1998, p 280-287

[52] F. Basegmez, “Extending a scientific application with scripting capabilities”, *Computing in Science & Engineering*, Volume 4, Nov/Dec 2002, p. 52-59