Interoperability of DEVS and C-BML Simulation Models

Elizabeth Hosang, Gabriel A. Wainer
Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada

Abstract - Simulation Interoperability allows systems that have been developed using different technologies on different operating systems to work together. The Joint Command, Control and Communications Information Exchange Data Model, Coalition Battle Management Language and Military Scenario Definition Language define formatted schemas that can be used to communicate information during a scenario. The Discrete Event System Specification (DEVS) is used to create models whose behavior is defined in response to events in the simulated environment. RISE provides a web-based interface for executing DEVS models. We present an application that uses formatted messaging to interact with a DEVS model running on the RISE server. The purpose of this is to demonstrate that a DEVS model can be executed as part of a larger, web-enabled synthetic environment.

Keywords: DEVS, RISE, CD++, C-BML.

1 Introduction

Computer simulation can be used to evaluate the behavior of systems, resources and perform planning for large-scale operations. Simulation allows planners to evaluate the behavior of complex systems without deploying and consuming the actual resources required in the scenario. Military organizations use simulation and modeling for mission rehearsal to evaluate planned courses of action. Simulation programs called Computer Generated Force (CGF) systems such as VR Forces and VBS2 [43] are used to allow commanders to develop mission plans and evaluate them without requiring the physical deployment of actual personnel and equipment to the field. These simulations may be used in training of new command staff personnel. From the point of view of the command staff, there needs to be no difference between the physical forces that are deployed in the field and the simulated forces, either when being tasked by command staff or when reporting back on progress or observations. [42].

Military organizations rarely operate in isolation. In order for their systems to interact, standards have evolved that can be used to allow the C2 systems to exchange data [42]. Coalition Battle Management Language (C-BML) is one such standard. It is based on the Joint Command, Control and Communication Information Exchange Data Model (JC3IEDM) which was developed by the MIP program as a basis for exchanging data. C-BML is designed to unambiguously communicate a commander’s intent. The Military Scenario Definition Language (MSDL) is a companion schema to C-BML, using the same syntax for initialization of systems prior to an exercise.

Civilian emergency response teams uses simulation for training and planning, although to a lesser extent. Most simulations involve first responder personnel responding to simulated emergencies with actors portraying victims in an emergency. Natural disasters such as floods and fires may require cooperation of military and civilian agencies. Planning for this type of large-scale operation could benefit from the use of M&S currently used by militaries.

In all these scenarios, the use of Web-enabled systems has evolved, bringing together more and more systems, including both military and civilian applications. As Web services become more complex and offer more services, the interfaces they present to users has become more complex. The Representational State Transfer (REST) architecture was introduced as a way to simplify the interactions of Web entities. The use of a RESTful interface on a service simplifies the interface of the service, thus simplifying the logic required in the client.

The Discrete Event Systems Specification formalism (DEVS) has gained popularity for modeling such complex systems. The RESTful Interoperability Simulation Environment (RISE) allows clients to access simulations using a RESTful interface. It hosts DEVS and Cell-DEVS models, allowing them to be run on behalf of a client. During execution the client can access the simulations using RESTful messages. In this paper we will explore the use of a web-enabled application written in C# accessing a DEVS model created using C++ on a RISE server. This demonstrated the successful interaction of a mix of technologies. Through the use of a scripted scenario, the application demonstrated how to generate initial state data using a structured format, a Report message of activities in the synthetic environment using a structured message format, querying a DEVS model on the RISE server for its response to the Report and parsing the results from the DEVS model to update the state of the modeled entity.

2 Background

High Level Architecture (HLA) is an architecture which allows multiple systems to coordinate. The individual systems, known as federates, coordinate by publishing formatted text messages on a Run Time Infrastructure (RTI), which manages the messages. The Base Object Model (BOM) [8] defines this common base. Each federate defines a Federation Object
Model (FOM) which must be understood by any other federate that needs to know about those objects. Distributed Interactive Simulation (DIS) 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 it defines Protocol Data Units (PDU) which specifies message formats in binary for communicating between simulations [1]. Binary data is more efficient to process than XML, but more difficult to understand.

While DIS and HLA are both standards meant to allow Command and Control systems to communicate, they are not immediately compatible. A DIS gateway may be used to allow a DIS model to communicate to an RTI. Alternatively, the Real-time Platform Reference Federation Object Model (RPR FOM) organizes attributes and interactions of DIS models into an HLA hierarchy [44].

The Joint Command, Control and Communication Information Exchange Data Model (JC3IEDM) are a logical data model that defines concepts that are common in a C3 environment. The model was developed by the Multinational Interoperability Program (MIP), a consortium of 29 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 [3]. Specialized functional areas such as fire support operations served as sources of requirements for the original development of the data [45]. The intent was to allow commanders to send instructions to units, and allow them to report back observed entities and their status. 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. Location definitions include points, sequences of points, polygonal lines, circles, etc [45].

The Coalition Battle Management Language (C-BML) standard is used to represent battle management doctrine in the Command and Control (C2) environment [1]. It was defined based on the JC3IEDM [2] and uses the category codes defined in that model. The C-BML definition is not a specific schema. Instead it is a set of building blocks that can be used to define custom schemas to be used during an exercise. Objects that are produced during an exercise featuring Command and Control are never updated or deleted. Instead, updates are created. For this reason they can be modeled as web resources. This makes C-BML suitable for implementation using RESTful web services [4]. 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 [6]. They have 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 [6]. The SBML provides a RESTful interface as well as a SOAP interface [7, 5].

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 organizational hierarchies, initial deployments, terrain and weather, and plan objectives. It is closely related to C-BML [1]. It reuses the Base Object Model (BOM) SISO 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 [5].

Data that resides in a fixed field within a record or file is referred to as structured data [46]. Unstructured data is data that does not reside in a traditional row-column [47]. The JC3IEDM, C-BML and MSDL schemas all define structured data. Because the structure and all possible values of the data are well defined this makes it possible for clients of this data to map the data to a format that can be understood by users.

Discrete Event System Simulation (DEVS) is a technique for modeling the behavior of systems as a series of behaviors in response to stimuli. The system is broken down into a set of one or more atomic models. Multiple atomic models can be coupled to form larger models [21]. The CD++ tool is a supports the definition and execution of DEVS models using C++. Each atomic model is defined as a subclass of the Atomic class. 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.

Representational state transfer (REST) is a style of software architecture for distributed systems. The intent of REST was to provide a simplified protocol using only four of the HTTP methods: PUT, POST, GET and DELETE. REST emphasizes scalability, generic interfaces, and independent deployment of components [9]. 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. RISE is a server with namespaces, arranged in a hierarchy. The root of the server is accessed 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. Models are executed by POSTing them to the RISE system under an appropriate namespace [30].

3 A RESTful C-BML environment

Discrete Event Simulation has been proven to be a versatile technique for modeling a variety of systems. The RISE
framework provides a RESTful framework for executing DEVS models. This interface provides the opportunity to integrate a DEVS model into a larger synthetic environment via simulation integration techniques such as the use of C-BML messaging. In this section we show how to build a proof of concept system to demonstrate the interoperability of a DEVS model running under RISE. This is done by using an external application to interact with a RISE server to execute a multi-part scenario.

In the Conceptual Architecture for this experiment, a DEVS model would execute in Real Time. Prior to the start of execution of the scenario, a DEVS Bridge would load the DEVS model on the RISE server, and supply the initial state information for the DEVS models. Once the simulation started, C-BML messages from other entities in the larger synthetic environment would be received by the DEVS Bridge, which executes as an independent process and can communicate using web-protocols. The DEVS Bridge would then forward the messages to the DEVS model on the RISE server, which would start execution of the DEVS model. Once the model had completed executing, the DEVS Bridge would detect the output from the DEVS model and publish it to the synthetic environment, either via the C-BML Server or an RTI. For the purposes of this scenario the DEVS model represents Emergency Services. The atomic models are a central Dispatcher which tasks a Police Car/Unit and a Fire Truck/Unit. The Dispatcher receives messages from the external DEVS Bridge, and determines the tasks that the units should perform. The Police and Fire Units determine the time at which they would start the task and passivate. Since the DEVS model uses a simulated clock the output from the DEVS model is generated right away and includes the time of the output. The output is then parsed by the DEVS Bridge and reported to the rest of the larger simulated environment.

The key conceptual entities are shown in the figure below.

![Diagram of Conceptual Entities in the Emergency Services Dispatch scenario](image)

The Dispatcher entity is responsible for responding to messages from external systems and sending Tasks to the Fire and Police units. However, in implementation it must also bridge the technology gap between the external systems and the DEVS model. Rather than assign the Dispatcher model two distinctly different responsibilities, a Gateway model was added. The Gateway does not participate in the simulated behavior of the system. Instead, its job is to parse incoming messages from the external system, and translate them into the data model used by the Atomic model. It also parses the initial data state information. The Gateway then distributes this information to the DEVS model, which triggers the execution of the DEVS model.

Because the intent of C-BML schema is to unambiguously communicate the commander’s intent, it is highly structured data and uses a number of category codes that are used to define the entities in the simulation, their locations, tasks, equipment, etc. However, the category codes which are used are those defined in the JC3IEDM schema. This schema was developed to direct the behavior of military entities. Where civilian entities are represented, they are represented incidentally, as the targets of activities or entities in the environment. Part of the motivation for this experiment is to explore integrating new simulations into a larger synthetic environment. For that reason civilian entities were selected for this experiment. However, there are few category codes in JC3IEDM that can be used to represent civilian agencies, and no category codes that could be used to task the civilian agencies modeled in this scenario. For example, there is no category code for fire-fighting units, nor is there an Action-Task code for “Fight Fire”.

The DEVS Bridge application uses a number of classes to manage the data used to generate the Report messages as input to the DEVS model. The data classes are also used to create the RESTful messages that are used to post the model and retrieve the results. The message classes are shown in Figure 2 below.

![Diagram of Message Classes](image)

The ResourceSet class contains all of the resources used in a given scenario. The class contains two lists: a list of UnitData instances and a list of LocationData instances. The DEVS Bridge maintains one instance of ResourceSet during processing. All initial state data is generated using the ResourceSet, and when the output from the DEVS model is parsed it is used to update the location and current tasks in the ResourceSet. In addition to maintaining the two lists of data, the ResourceSet class also provides a number of methods that
support the use of the class and hide its internals from users of the data. The key responsibilities of the class are to maintain the data, and to support users iterating through the list.

The MessageSet class contains the Report messages used in a given scenario. The class contains a single list of MessageWrapper instances, each of which has a CbmlMessage as its MessageBody attribute. The entire collection can be serialized into or deserialized from XML format using the SerializeUtility. This is done to save the messages to file, and to read them back into the DEVS Bridge from file. The MessageWrapper class holds information for a single message. It can be used for RESTful messages that are sent to the server, or for Report messages which are sent to the executing DEVS model. The responsibilities of the MessageWrapper class include methods for formatting the message, and for generating an HTTP message.

The CbmlMessage contains an ActionEvent and a LocationId. These are used to generate the Report which is sent to the DEVS model. Instead of generating a serialized message based on the attributes of the class, the message string is used as-is. This message type is used to generate the RESTful messages to create, post, execute, and delete the model, and to get the results or error files. The ReportMessage class holds the data that is used to generate all input for the current MA file, including a single MessageWrapper for the Report and the ResourceSet for the initial state data.

The Messages and Resource Set editors display data in a table form. This data is displayed using the standard Data Grid View control supplied as part of the .NET Windows Form toolkit. Data Grid Views provide support for formatting displayed data, re-arranging columns, hiding and showing columns, and selecting rows or columns in the display. Data Grid Views require a DataSource to supply the data in the table.

The DEVS Bridge form has two main display areas: an area describing the scenario information, and a text window where the results of running the tool are displayed. The fields in the top half of the form are:

- **RISE Server** – address and directory where the model will be created/posted.
- **Model Name** – name of the model that will be used to create the model on the server. The string “.Model” is appended to the text in the Model Name field. The value of this field may be entered manually or will be set after zipping the model. This is done by selecting the “Model Prep” option under the “Execute” menu. This function zips up the DEVS model files and generates the XML file required to create the model on the server.
- **Source Directory** – the location of the zipped model that will be uploaded to the server. This value is also set after executing the “Model Prep” function. It can also be edited manually or the browse button to the right of the field (“…”) can be used to locate the directory where a zipped model has previously been stored. This value is appended to the RISE Server address on the main DEVS Bridge window.
- **Message File** – the name of the file containing the loaded message set for the scenario. This may be blank if no file has been loaded or saved. The Execute button is not enabled if no messages have been loaded.
- **Resource File** – the name of the file containing the loaded set of resources (Locations and Units) that are used in the scenario. This may be blank if no file has been loaded or saved. The Execute button is not enabled if no resources have been loaded.
- **Next Report** – a description of the next Report message to be processed by the DEVS model.

The Status text pane at the bottom of the window displays output from operations performed by the application. An example of the output in the Status window is shown in Figure 4 below.

![DEVS Bridge Main Form](image)

**Figure 3 DEVS Bridge Main Form**

The DEVS model provides a simplified model of a civilian emergency response management system. It represents a centralized Dispatch office, such as a 911 operator, which receives reports of an Action Event and assigns an Action Task to multiple types of responders, such as Police and Fire.

![Status window displaying Messages and Resources](image)

**Figure 4 Status window displaying Messages and Resources**

The DEVS Bridge form has two main display areas: an area describing the scenario information, and a text window where
The Gateway Model represents the interface between the DEVS model and external simulations. It is responsible for receiving data from the DEVS Bridge and parsing it into the internal data format used by the atomic models.

The initFunction method loads the event, units and locations strings from the MA file and passes them to the MessageHolder instance to populate it. After all of the models have initialized, the Gateway model sends the ActionEvent and a Location Id to the Dispatch model.

The Dispatch Model collects the Action Event and Location Id from the Gateway. When it has them both it examines the Action Event. It determines which tasks to assign the Police and Fire units based on the Action Event. It sends the Action Tasks and Location Id to the Police and Fire units. It does not need to load initial state data.

The Police Unit model loads its initial state information from the MessageHolder on the Gateway. When it receives its task and a location id from the Dispatch model it queries the MessageHolder for the details of the Location. If the Task is ReturnToBase, the Police Unit requests the location of its home base. The model then calculates the time required to travel from its current location to its next location.

The Fire Unit model loads its initial state information from the MessageHolder on the Gateway. When it receives its task and a location id from the Dispatch model it queries the MessageHolder for the details of the Location. If the Task is ReturnToBase, the Fire Unit requests the location of its home base. The model then calculates the time required to travel from its current location to its next location.

The Action Task enumeration class corresponds to the Action Task category code defined in the DEVS Bridge project. It is used to send tasks to the units. The enum value is cast as an integer when it is sent to the out port. In the externalFunction on the Dispatch class the msg.value() is cast to the enumerated type.

The Location class represents a single location. It stores the unique location id for the location and the latitude and longitude of the location. The UnitRecord class represents the state of a unit. It holds the unique location id for the unit, the unique id for the unit’s location, and the current ActionTask of the unit.

The Gateway passivates for a few milliseconds, but for a longer time than the PoliceUnit and FireUnit classes. This is done so that it does not send any output until after the classes have loaded their initial values. When the internalFunction executes the Gateway exports the location and action event data to the Dispatch class. The sequence of behavior is shown in the diagram below.

The main functions are shown in the figure. Atomic Models do not invoke each other directly. The sending model invokes sendOutput and passes in the name of the port, the value of the message, and the time. The externalFunction method is invoked on the receiving model. However, for simplicity the diagram shows the sending model invoking the externalFunction method on the receiving model. At the end of the externalFunction and internalFunction methods the model either calls the passivate or the holdIn method. However, the diagram only shows the holdIn methods on the PoliceUnit and
FireUnit at the ends of the sequences. These methods are shown because they represent the calculation of the time required for the units to arrive at their new locations. When the holdIn method ends the units send their new task and location id information to their output files. The time, task, location and output port information is written to the out file.

The DEVS Bridge was implemented and tested with the DEVS model. The results were successfully parsed and used to generate new messages. The DEVS Bridge was also used to post a pre-existing model to the server and download the results, proving that the tool can also be used as an interface for posting models to the server.

4 Conclusions

We showed the use of a web-enabled application written in C# accessing a DEVS model created using C++ on a RISE server. This demonstrated the successful interaction of a mix of technologies. The application shows that Discrete Event Simulations running under the RISE system can participate in a simulation with other systems that are coordinated by exchanging C-BML messages, using subscriptions and notification to be alerted to the presence of data. The application demonstrates a proof of concept system showing that a DEVS model executing on a RISE server can participate as a part of a larger synthetic environment with the use of a bridge application that acts as a front end to the RISE system.

A custom set of Action Event, Action Task and Unit Type category codes were defined. This issue is one of the reasons why a custom message format was created. This issue cannot easily be addressed. The C-BML schema is based on the JC3IEDM definition. Both are defined by international committees based on the requirements of military organizations and the JC3IEDM model has been in used for many years.

If civilian agencies are to be modeled in simulations that use C-BML for interoperability, they will have to work around the current schema definition. There are several possible options for this:

- Use the “Not Otherwise Specified” / NOS value defined as a value in most category codes.
- Extend the schema and define custom category codes, or extend the existing ones.
- Select existing category codes and map them internally to appropriate values for the simulation. For example, sending an “ATTACK” tasking to a Fire Unit could be interpreted as performing their main function, such as “FightFire”. However, this goes against the principle of unambiguously communicating the commander’s intent.

5 References

[1] An Architecture for Demonstrating the Interplay of Emerging SISO Standards; Blais, Gustavsson, Gustavson, Reichenthal, Fall Simulation Interoperability Workshop (FallSIW): 10-15
[5] Lessons learned from NMSG-085 CIG Land Operation demonstration, Gautreau, Khimeche, Remmersmann, Martinet, Muniz, Serrano, Pedersen, Lillesoe, Henderson, Liberg
[12] Adding Publish/Subscribe to the Scripted Battle Management Language Web Service; Corner, Pullen, Singapogu, Bulusu
[13] Performance and Usability Enhancements to the Scripted BML Server. Pullen, Corner, Nicklas, C4i Center, GMU
[14] Coalition Battle Management Language (C-BML) Phase I Standard: Trial Use Findings and Next Steps; Blais, Alstad
[15] Lessons learned from NMSG-085 CIG Land Operation demonstration, Gautreau, Khimeche, Remmersmann, Martinet, Muniz, Serrano, Pedersen, Lillesoe, Henderson, Liberg
[17] Maturing Supporting Software for C2-Simulation Interoperation Pullen, M and Nicklas, L, C4I Center, George Mason University, Fairfax VA, USA


[32] Zhan Qian; Wei Xianglong, "The research and implementation of a RESTful map mashup service," *Second International Conference on Communication Systems, Networks and Applications*, pp.401,4032010


[45] The Joint C3 Information Exchange Data Model (JC3IEDM Main), 14 February 2012

