Methodology

JDMS

Journal of Defense Modeling and Simulation: Applications, Methodology, Technology
1–23

DOI: 10.1177/1548512915607659
dms.sagepub.com
SAGE

An architecture to facilitate interoperability of Discrete Event System Specification and Coalition Battle Management Language simulation models

Elizabeth Hosang[1,2] and Gabriel A Wainer[1]

Abstract

The Joint Command, Control and Communications Information Exchange Data Model, Coalition Battle Management Language and Military Scenario Definition Language define formatted schemas that have been developed for use in military simulations. The Discrete Event System Specification (DEVS) is used to create models whose behavior is defined in response to events in the simulated environment. The RESTful Interoperability Simulation Environment (RISE) provides a web-based interface for executing DEVS models. DEVS models can be used to simulate systems as diverse as natural disasters and traffic patterns. 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, such as a military planning exercise.

Keywords

Discrete Event System Specification, RESTful Interoperability Simulation Environment, CD++, Coalition Battle Management Language

1 Introduction

Simulation software is being used more often 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 behavior 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 behavior of biological systems, from cancer cell growth to insect population behavior.[2] Using simulation and modeling allows researchers to control external factors. Simulations can also be used to model natural disasters, such as floods and forest fires. Execution of the simulation allows responders to predict the behavior of the disaster and evaluate possible courses of action.[2] The ability for simulations to interact allows the building of scenarios that are more complicated by having multiple scenarios working together.[1] One simulation system may model the behavior of troops, while another may model the behavior of civilians in the area. Different simulation systems can be used to model a wide range of scenarios. One method that has gained popularity in recent years is the Discrete Event

[1]Department of Systems and Computer Engineering, Carleton University, Canada
[2]CAE Canada, Canada

Corresponding author:
Gabriel A Wainer, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada.
Email: gwainer@sce.carleton.ca

System Specification (DEVS) formalism.[2] DEVS formalism allows systems to be modeled as a series of discrete events occurring over time. Cell-DEVS is an extension of the DEVS formalism that can also be used to model systems that can be defined as cellular models.

There are multiple simulation systems available, and in order to make them interoperable, numerous standards have emerged, such as the High Level Architecture (HLA)[3] and Distributed Interactive Simulation (DIS)[4] standards. In these standards each component in a simulated environment can be modeled by a different system. This allows the simulation to leverage the distinct strengths of each system to allow simulations to cooperate across multiple machines and computer domains.[5] In this sense, the Coalition Battle Management Language (C-BML)[6] is an Extensible Markup Language (XML)-based specification that defines elements that can be used to create reports about simulated entities and their behaviors. C-BML 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 C2 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.[7] 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 C2 software with different types of simulation systems to execute a large exercise in a single synthetic environment.

Likewise, web protocols represent another technology that is used to communicate between multiple systems. 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 remotely that uses a different operating system. The popularity of web protocols for communications has led to them being used to communicate between simulation systems as well as real systems.

In order to make it more accessible for interaction with other systems, the RESTful Interoperability Simulation Environment (RISE) server was developed.[9] The RISE server provides lightweight middleware for executing DEVS models. It provides an Application Programming Interface (API) that allows modelers to create DEVS models. This interface allows the DEVS capability to interact as a part of a larger simulation. The Representational State Transfer (REST)-ful 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.

The goal of this research is to define an architecture that 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 is analyzed by creating a synthetic environment that 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.

We also want to examine the suitability of C-BML for sending reports of events in the simulated environment. Another goal of this research is to examine the use of MSDL to initialize DEVS models running on the RISE server. This research 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 research is to examine the C-BML and MSDL standards for their suitability for use describing the interactions of civilian entities. Emergency scenarios, such as fires and floods, also involve civilian emergency service agencies, often working in concert with military organizations. The C-BML and MSDL specifications will be examined for their usability in modeling civilian entities and this research will present the results of this study.

We also want to check the validity of the proposed architecture. We show a case study that models the behavior 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.

2 Background

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 scenarios that are more complex 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. Following, we will discuss some of these standards.

The HLA standard[1] provides a communication infrastructure to allow simulations to coordinate via formatted messages. 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.

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. The use of FOMs over a number of years allowed common attributes to be identified as being of use for all objects in a FOM, regardless of the type of environment being modeled in the federation. This led to the development of the BOM.[7]

The BOM is a conceptual model of a simulated system. It facilitates interoperability, reuse and composability by providing a common set of basic attributes.[5] 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.

The 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 (PDUs) that specifies 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 the HLA, where updates are delivered only to simulations that subscribe to the RTI.

While DIS and HLA are both meant to allow C2 systems to communicate, they are not immediately compatible due both to the format in which data is exchanged, and to 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 FOM and publishes them on the RTI. The Gateway also performs mapping from HLA to DIS. To standardize the behavior of Gateways, the Real-time Platform Reference Federation Object Model (RPR FOM) was defined. This FOM organizes attributes and interactions of DIS models into a HLA hierarchy.[10] While it was defined for use in Gateways, it can also be used in federations that consist only of HLA-compliant federates.

The Joint Command, Control and Communications 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 29 NATO and non-NATO nations, in order to promote international interoperability of C2 Information Systems.[11] The JC3IEDM was initially designed to capture data required for C2 operations. Specialized functional areas, such as fire support operations, served as sources of requirements for the original development of the data.[12] 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.[12] 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. 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 that are observed being performed by entities outside the control of the commander, such as civilians, hostile forces and non-governmental organizations (NGOs). In contrast to JC3IEDM, where data is exchanged by database, the C-BML standard is used to exchange battle management doctrine in the 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[6] and uses many of its Category Codes directly.

C-BML is not a specific schema: it is a set of building blocks used to define message formats for 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 C-BML capture the five *W*s of information: *who, what, when, where* and *why.*[6] The standard uses many of the entities defined in the JC3IEDM, 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.[6]

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, for example unit should move *x* amount of time after an event, or absolute time, for example an event occurred at time *x*.
- Who – describes actors performing an activity.
  - The Who entity may represent:
    - an organization – a group of persons, such as a unit, a company, an army, or a NGO;
    - 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, for example an 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 C2 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 RESTful web services.[13]

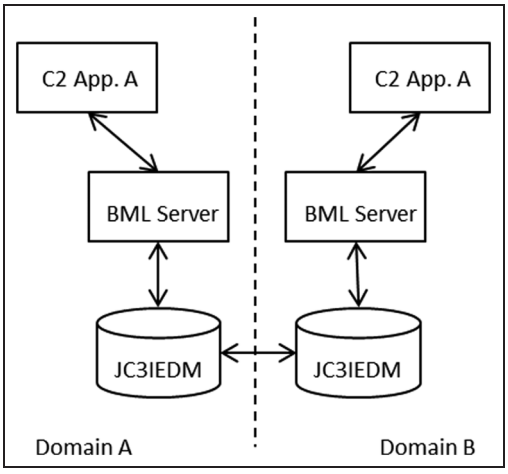

Figure 1. Typical Coalition Battle Management Language (C-BML) server deployment.

The C-BML server configuration is shown in Figure 1. 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.

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.

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.[13] A Use Case for an initial demonstration of using multiple specifications was given by Blais et al.[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 harbor. The vehicles included unmanned surface vehicles (USVs), unmanned aerial vehicles (UAVs), unmanned undersea vehicles (UUVs) and unmanned ground vehicles (UGVs). 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.[13]

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.[14] 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 CGF application.

Instead of using the full JC3IEDM database storage simple XML file storage was used, which reduced the maintenance costs of the Systems Biology Markup Language (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.[14]

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.[14] Simulation has been used to investigate asset utilization and planning when the military has been deployed as backup in the case where fire fighters' unions went on strike.[15] This type of simulation has become more important since the events of September 11, 2001.[16] Simulation and other models are being used for emergency management, as documented by Tao et al.,[17] and to simulate natural disasters.[18] Simulation is also being used for aspects of health-care modeling, including the management of Emergency Medical Services (EMSs)[19] and triage simulation.[20]

Besides the underlying infrastructure for simulation and interoperability, it is important to have a good mechanism for modeling. The DEVS is a technique that can help in modeling the behavior of these systems. A DEVS model 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 before an internal transition generates the reaction of the system. Multiple atomic models can be coupled to form larger models.

The CD++ tool is a DEVS simulation engine with 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 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.[21] DEVS has also been used to build testing and evaluation systems that can be accessed via web technologies.[22] DEVS simulations have been used to evaluate the behavior of modeled systems, such as cellular networks,[23] model biological systems, physical systems, etc.[2]

To make the DEVS models available to military synthetic environments, a method is required to allow the different technologies to interact. 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.[24] The intent of the REST style was to provide a simplified architecture. REST emphasizes scalability, generic interfaces and independent deployment of components.[24] Instead of defining a protocol that can accommodate a growing set of message types, the protocol uses only four Hypertext Transfer Protocol (HTTP) message types: PUT, POST, GET and DELETE. All operations performed by the web service are modeled as changes to resources defined on the service. The resources are identified through a Universal Resource Indicator (URI) that 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 Uniform Resource Identifier (URI) for the resource. Additional information may be included in the HTTP message body.

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.[25] Similar differences in sensor types led to its being used in other web sensor networks.[26] 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.[27] The simplicity of the REST interface compared to the traditional SOAP (Simple Object Access Protocol) interface has led to efforts to access current services via a simpler interface, for developing a semantic web.[28] SOAP allows users to build complex applications providing interoperability under various middleware. Combining SOAP with WSDL (Web Services Description Language) and XML Schema allows defining advanced services on the web. XML provides a standard mechanism for defining these services. Nevertheless, Web Services definition through SOAP is more complex to interoperate with simulation purposes. RESTful techniques are also being used as part of efforts to develop mashups of existing services, as described by Tosic and Manic,[29] Pan and Liang[30] and Bo et al.[31]

RISE[9] follows these ideas, providing a framework for executing distributed simulations, and it has been thoroughly tested for DEVS. 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 built as a RESTful web services server with namespaces, arranged in a hierarchy. The root of the server is accessed via the 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. Models are executed by POSTing them to the RISE system under an appropriate namespace.[32] RISE 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.[33]

3 Architecture for a distributed simulation using the RESTful Interoperability Simulation Environment and Coalition Battle Management Language

In this section, we present our architecture, which 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 behavior of simulated entities, or provide information that affects the behavior 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. 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).

We propose a way to achieve the second capability by defining a new service: the DEVS Bridge, which 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, for example current location of simulated entities.

The proposed architecture is shown in Figure 2. For the sake of simplicity, the workspaces on the RISE server are not shown.

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

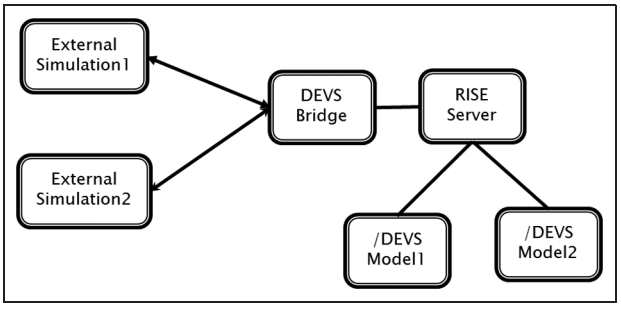

Figure 2. Architecture overview.

other federates, or could subscribe to an operational database using the JC3IEDM schema, receiving notifications of new information and publishing the results of the DEVS model execution to the database. For the purposes of this research 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 research is designed to receive information as XML-formatted messages that are to be parsed, translated and presented to the DEVS model as input.

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.

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 the 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 in Figure 3. The diagram assumes that the scenario is being controlled by one of the external simulations.

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

- The external simulation that 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.

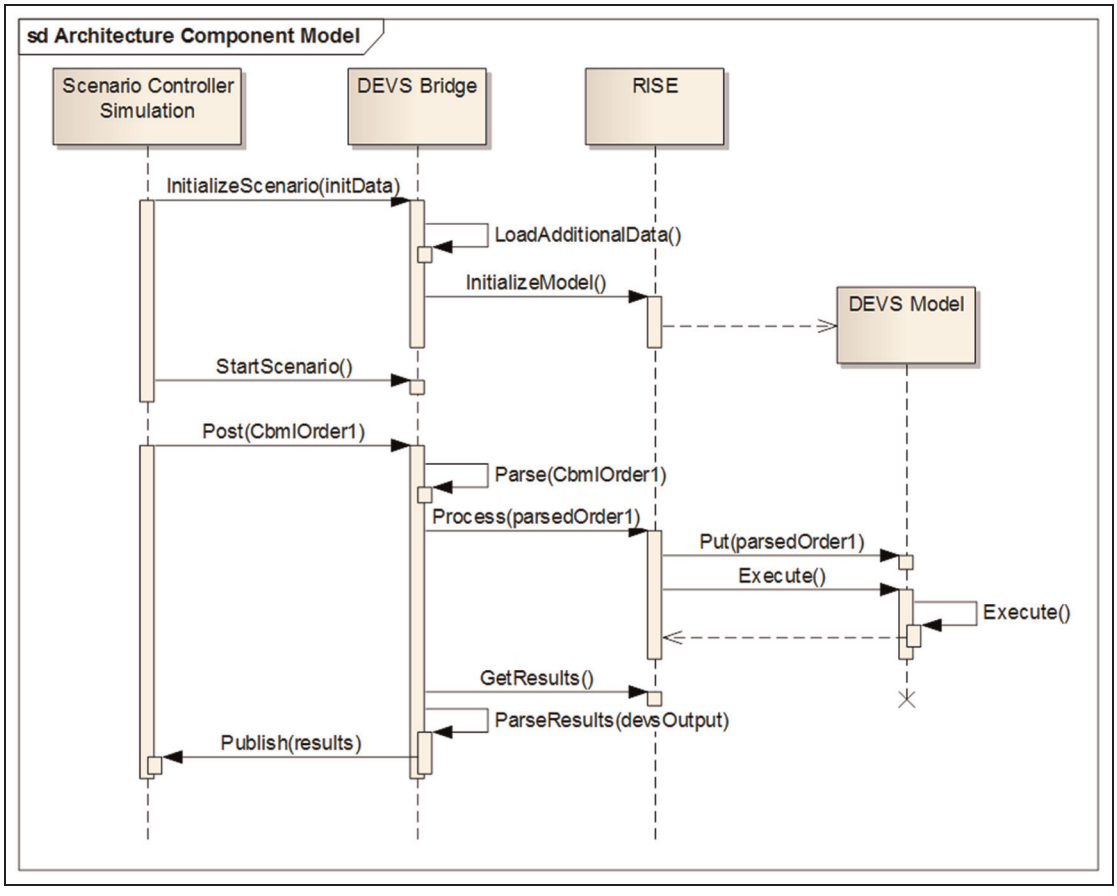

Figure 3. Messaging between components at the architectural level.

- 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 parts of this section discuss the DEVS Bridge design and capabilities in more detail.

3.1 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, which are required in order to ensure the smooth operation of the DEVS model during the simulation.

The execution of a model on RISE runs from a given simulated start time to an end state, which provides information to the larger simulation about the entities modeled by the DEVS model. In response to this information, the entities in the larger simulation may take specific actions, resulting in the need to execute the DEVS model again to determine the new behavior of its modeled entities. Between these two executions of the DEVS model, the state information describing the entities modeled by the DEVS model needs to be preserved so that it can be provided as input to the next execution of the DEVS model. For this reason, the DEVS Bridge needs to be able to manage the state information for the entities of the DEVS model, and the DEVS Bridge must 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), etc. 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 if the 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 Bridge determines that it has received a message pertaining to the entities managed by the model, it needs to generate all required inputs. This includes formatting state information messages so that all of the atomic models are 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 Bridge. After the model has executed, the 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.

RISE provides an environment for creating a workspace for a DEVS model and executing it. However, this model must reside on the server before it can be executed. The location of the model must be known to the Bridge. Therefore, the Bridge has the capability to create the workspace on RISE, post the model to the server and get the execution results. All general maintenance operations required to manage a model on RISE 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.[33]

3.2 DEVS Bridge design

The DEVS Bridge was designed with a number of capabilities that are required to execute a scenario. These include model creation, message set management and state management. 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 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 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.

In addition to a set of messages, the DEVS Bridge requires a set of resources in order to execute a scenario. The Bridge was designed with the capability to manage the resource set. Before the scenario can start, the Bridge must have loaded a set of resources, which 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, 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 Units. The Location Editor and Unit Editor allow the creation of new items or the editing of existing items. Because Units have locations, 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. 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.

3.3 Use of C-BML messaging specification

The C-BML specification was developed to work as part of a larger synthetic environment where messaging between simulations is done via a message format defined using C-BML. As described earlier, 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 organizations. It therefore contains elements designed to convey information about troops, equipment, terrain, installations and weather conditions. Specifying information about a single unit requires specifying separate entities for the location of the unit, the type of the unit and the time at which the unit is scheduled to perform an action. Any equipment held by the unit, such as vehicles, weapons and munitions, also requires separate entities. The specification also defines all possible values for the attributes of all entities. The entities are tied together with unique identifiers for the entities.

All of this information is required to convey information unambiguously. 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.[13] However, the goal of this research is 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 research 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.

Because of the complexity of using C-BML-compliant messages, a simplified message schema was developed. The message schema supports the entities and messages defined for emergency services. The DEVS model represents emergency service units and a dispatch model that tasks the units. The units are dispatched based on emergencies that are modeled by the external simulation. The external simulation supplies reports of events to the DEVS model, which then responds by calculating the behavior 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 the start time of the scenario. Therefore, the report schema need only describe the *what* and *where*.

3.4 Use of MSDL model state information specification

C-BML is used to define messages sent between simulations working together in a larger simulation, to exchange 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 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 Object Identifier (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 organizations involved in the scenario, including organizations and force sides. It defines logical overlays used to group intelligence elements in the 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.

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.

3.5 Data Model

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 and 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. C-BML 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 Military Grid Reference System (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 Unit Class describes entities that participate in a scenario. It is a simplified version of the Unit-Object-Item defined by C-BML.

The Unit Type field replaces the Unit-Object-Type defined by C-BML. 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 the following:

- a new Action Task for the Unit to execute;
- a new Location for the Unit; and
- the time it will take 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 as follows:

- Report ID – unique identifier for the Report Message;
- Action Event – a value from the set of valid Action Events for the model;
- Location – unique ID of the Location at which the Action Event takes place.

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 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—Discrete Event System Specification Bridge and Discrete Event System Specification model

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 earlier, 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 modeled 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 as follows:

- 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 DEVS Bridge component has the following capabilities:

1. loading the Message and Resource Set files required for the scenario;
2. maintaining the current state of the entities in the scenario (units, locations) between executing messages in the scenario;
3. generating the Report Message in the DEVS-native format;
4. generating the current model state description in the DEVS-native format;
5. interacting with the RISE server to post and execute the DEVS model;
6. stepping through the set of messages to execute the scenario; and
7. displaying status information and response data from messages sent to the RISE server.

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 for the system being simulated. 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.

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.

4.1 DEVS model discussion

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.

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.[34] 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.
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. The data model details will depend on the scenario being modeled. However, some general guidelines 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.

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 behavior 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 behavior of the entities one step at a time, where one step is the behavior 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

For the purposes of validating the architecture, a DEVS model was required. The selected scenario models the C2 of emergency services. As discussed earlier, 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 organizations 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 modeling using a Discrete Event simulation.

In the real world, in 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 is no way to 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 Figure 4.

The police and fire departments have their own dispatch services, but for the sake of this scenario, only a single dispatch entity is modeled. The scenario that was modeled 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 as follows:

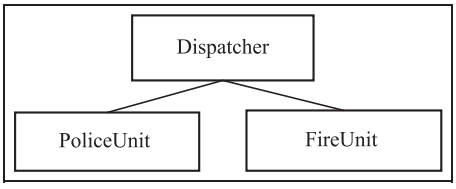

Figure 4. Conceptual entities in the emergency services dispatch scenario.

- 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 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 modeled entities.

The actual behaviors 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 given below and in Figure 5.

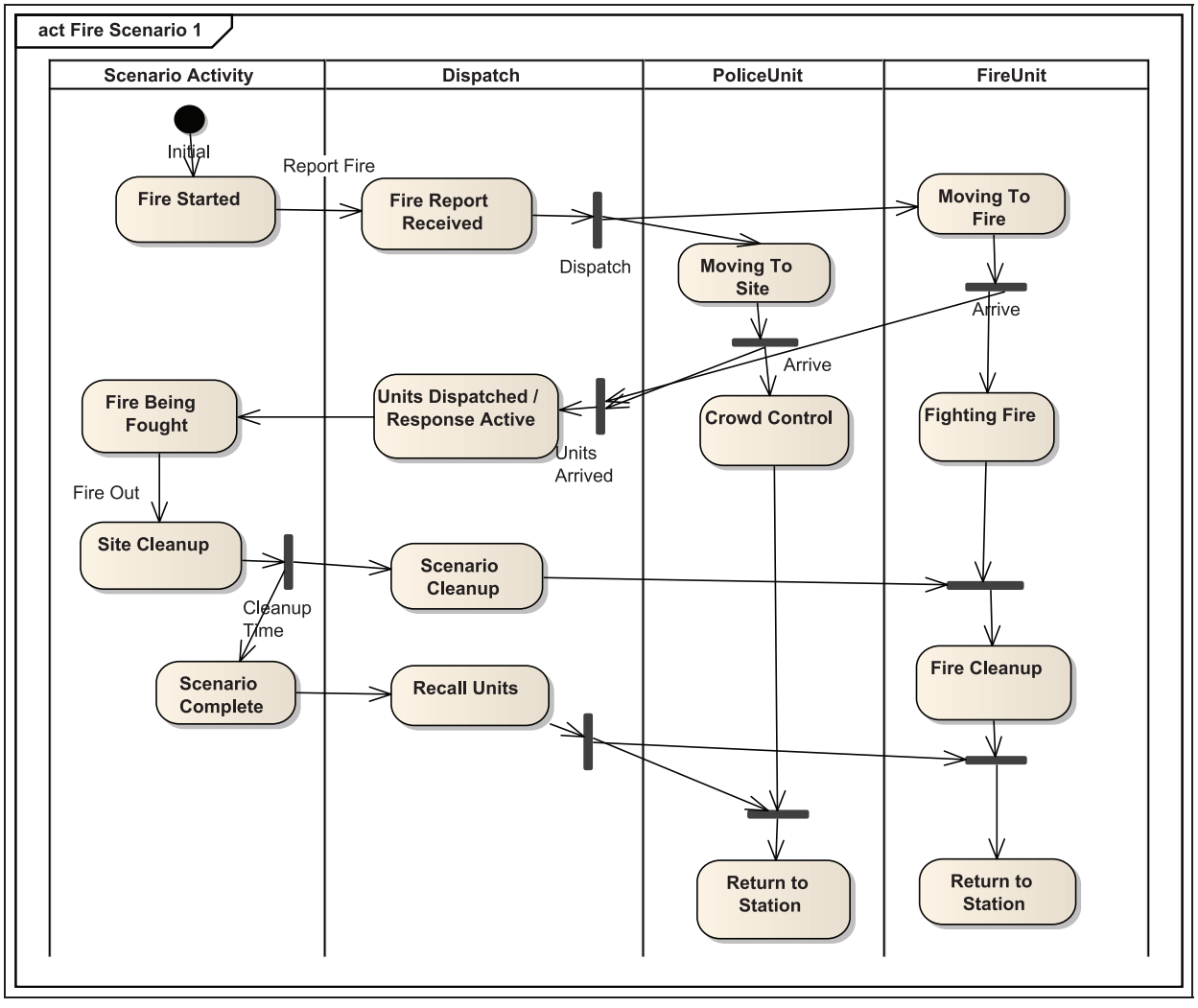

Figure 5. Conceptual activity sequence in a basic scenario.

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 that 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 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 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 (PoliceUnit) – the police car returns to the police station, or otherwise resumes its routine activities.
17. Return to Station (FireUnit) – the fire truck(s) return to the fire station.

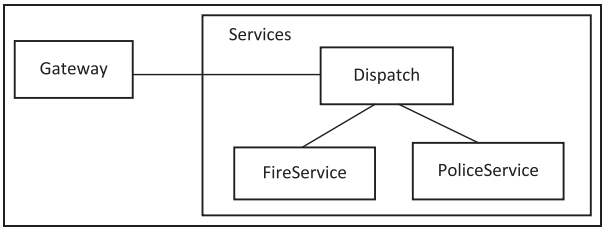

Figure 6. Discrete Event System Specification models.

These behaviors 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 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, PoliceUnit and FireUnit. A Gateway model is also required. The entities described in the figure become the atomic models in the DEVS model of the emergency services, which is used to validate our architecture.

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 Figure 6.

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.

The Gateway atomic model coordinates the behavior 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; MessageHolder contains a list of Locations and Units that are populated by parsing the state description information. MessageHolder also contains the Report Message. The MessageHolder instance is populated by the Gateway model. The classes are shown in Figure 7.

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

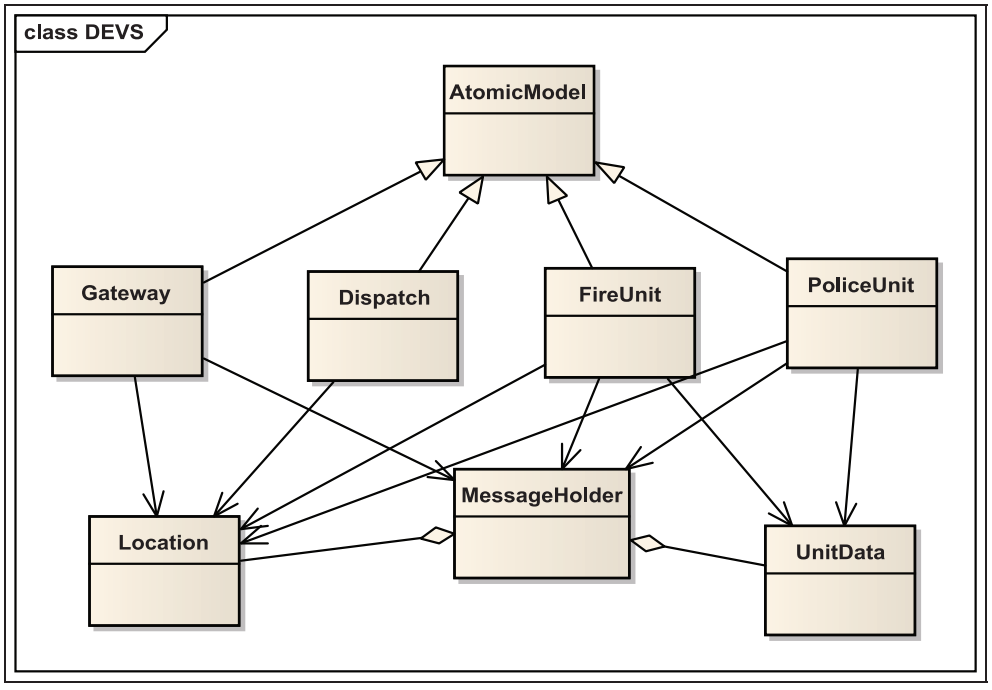

Figure 7. Discrete Event System Specification model class diagram.

during the execution of the scenario. The Report Message and initial state description information are included in the model definition 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 PoliceUnit and FireUnit internal transitions that load the initial data. 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.

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, they appear in an output file. 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 Table 1.

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.

Table 1. Action Tasks assigned by dispatch based on Action Events.

Action Event	PoliceUnit task	FireUnit task
Fire in Progress	Crowd Control	Fight Fire
Fire Out	Crowd Control	Clean Site
Emergency Over	Return to Base	Return to Base

The behavior of the Dispatch model is not dependent on any initial state information. It does not request any data from the MessageHolder on the Gateway. The FireUnit entity is responsible for responding to instructions from the Dispatcher. The instructions from the Dispatcher often require the FireUnit to move to a new location. The time at which the FireUnit arrives at the new location depends on the location of the FireUnit when it receives the new instruction from the Dispatcher. The FireUnit 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

PoliceUnit when it receives the new instruction. The behavior 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 coordinate 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 coordinate 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.

5.2 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 model definition of 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.

Once the DEVS model input has replaced the contents of the DEVS coupled model found in the coupled model 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. 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.

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.

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. 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 an output file generated by the DEVS model. The 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 that 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 output file, including the time at which the output is produced.

A sample of the output file from the DEVS model is shown in Figure 8.

The first four messages are produced by the internal transitions by the PoliceUnit and FireUnit when they load their initial state information, including their current location and task. These operations are scheduled a few

```
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 8. Sample Discrete Event System Specification model output.

Table 2. 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

Table 3. 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

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 begin to perform their new task. This output was produced in response to the initial Report Message of the fire in progress. Both the PoliceUnit and FireUnit report their new location as 300, which is the location of the fire. Their different tasks, 4 and 3, correspond to Fight Fire 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.

5.3 Other simulation scenarios

In addition to the basic scenario described above, we show two additional scenarios. Firstly, a second fire unit was requested to report to the scene to fight the fire. Then, 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 2. 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.

In order to add a second fire unit to the scenario, we made a number of modifications to the DEVS model described above. New ports were added to the Dispatch model to communicate with the second fire unit. The model definition file was updated to define a second instance of the FireUnit atomic model and connect it to the new ports on the Dispatch model. Additional output ports were so that outputs from the second fire unit would be added to the output 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 3.

Table 4. Dispatcher tasking of ambulance unit.

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

The following 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 behavior 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 behavior of the dispatcher was modified to issue these new tasks. The new behavior of the dispatcher is shown in Table 4.

The dispatcher does not assign any tasks to the ambulance unit for the FireInProgress or FireOut Action Events. New ports were added to the dispatcher to send data to the ambulance unit. The model definition 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 output file. The messages in the third scenario are shown in Table 5.

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 to 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 shopping mall 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 shopping mall 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 Table 6.

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 Table 7.

The ambulance is located at the same fire station as the second fire unit, Truck2. When calculating the time required while performing 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 coupled model file.

Table 5. 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

5.4 Case study results and analysis

A sample of the results of executing the messages is shown in Table 8. The times listed correspond to Resource Set 3.

In each of the three scenarios, the runs with the three different sets of locations resulted in different times of arrival of the units 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 sequences 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.

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. Once the C-BML message has been parsed and sent 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.

Table 6. Locations for resource sets.

		Coordinates		
Unique ID	Location name	Resource set 1	Resource set 2	Resource set 3
100	Shopping Mall	(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)

Table 7. 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

Table 8. 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 Shopping Mall	Crowd Control
00:06:00	00:07:00	00:07:00	Truck1 arrives at Shopping Mall	Fight Fire
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 Shopping Mall	Treat In Place
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 Shopping Mall	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	Return To Base

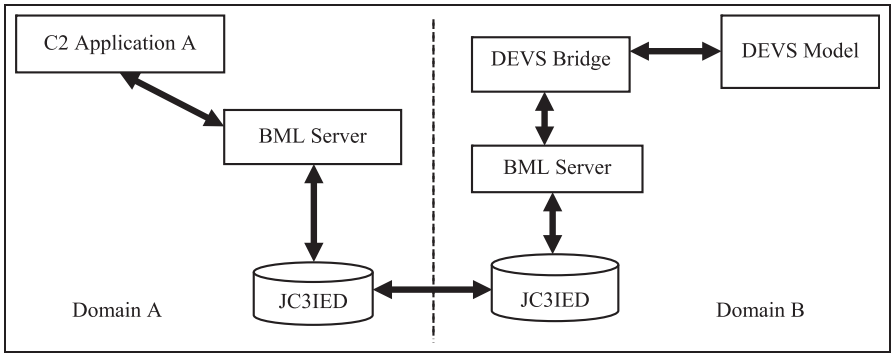

Figure 9. Discrete Event System Specification (DEVS) Bridge communicating via the Battle Management Language (BML) server and database replication.

One possible configuration for allowing simulations to communicate with each other in a larger environment is shown in Figure 9.

In this version of the configuration, the entities in Domain B are modeled 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. 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 modeled 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.

5.5 Architecture discussion

Simulation interoperability is categorized by the level at which the simulations are compatible. The Levels of Conceptual Interoperability Model (LCIM) defined by the Virginia Modeling, Analysis and Simulation Center (VMASC)[35] defines a number of levels that can be used to describe the way two models communicate. The levels range from Level 0, stand-alone systems, to Level 6, Conceptual Interoperability, where the models are fully specified, but implemented independently. The proposed DEVS Bridge uses C-BML and MSDL, which are intended to be unambiguous in meaning. This corresponds to Level 3, Semantic Interoperability. The meaning of the data is shared between the models using these protocols. At Level 2, Syntactic Interoperability, there is a common data structure to exchange information, but the meaning of the data is not necessarily specified. At Level 4, Pragmatic Interoperability, the simulations are aware of methods and procedures used by other systems. This is a greater level of interoperability than is provided by the DEVS Bridge, and would introduce a higher level of coupling between the DEVS model and the other models in the synthetic environment. This higher level of coupling would not provide any additional benefit to the simulation.

A number of simulation interoperability efforts have been made involving defense simulation tools interacting with other tools that have been designed specifically for military synthetic environments. The work presented by Ryan and Oliver[36] integrates UAVs. The results introduced by Roos et al.[37] look to validate and extend the support for maritime entities. The article by Mifsud et al.[38] is a discussion of the findings of exercises that combined military simulation tools such as Joint Semi-Automated Forces (JSAF) using Data Distribution Service DIS for data interoperability.

Modern military doctrine is based on the concept of full spectrum operations. Military organizations may be called upon to perform offensive operations, defensive operations and stability or civil support operations all in the same theater of operations.[39] The US Army's response to Hurricane Katrina is one example of the military supporting civilians and working with civilian agencies.[40]

This range of operations is also reflected in the standards published by NATO for governing the work of coalition forces. These Standardization Agreements, or STANAGs, cover a wide range of situations, from Munitions Management, STANAG 4629, to management of mass casualties, STANAG 2879, and medical support in environments where chemical or biological agents have been deployed, STANAG 2873. Other interoperability standards, such as C-BML and the JC3IEDM specification it is based on, include numerous category codes that describe non-military personnel, from journalists to village elders.[13] Planning exercises could benefit from the inclusion of simulations modeling non-military environments. In particular, DEVS and Cell-DEVS have been used to model traffic flow in urban areas,[41] evacuations of buildings and aircraft[42] and geographic features, such as forest fires and floods.[43] Adding these simulation features to military planning exercise would enhance the synthetic environment.

Development of the DEVS Bridge was done as a proof of concept to demonstrate that the DEVS formalism could be used with a structured communications protocol, such as C-BML. The transformation from the model represented by C-BML to the emergency services model was done at a very high level. The C-BML Object-Item entities were mapped to atomic models. The relationships between the entities were reflected in the Coupled Models. The Action Tasks from C-BML were mapped to the external events that triggered behavior in the overall model. Transformation from C-BML to a more complex DEVS model would be one area for future investigation. One approach would be to use Unified Modeling Language (UML) as an intermediary step in the transformation. Some efforts have already been made to map DEVS into UML.[44,45]

The C-BML and MSDL models have been validated in a number of scenarios.[13,46,47] The DEVS Bridge model developed for this paper was deliberately simplified. An area of research going forward would be to produce a model validated against the requirements of an actual emergency response system. The input and output of the DEVS Bridge is the external events sent to and generated by the DEVS model running on the RISE server. Validating the DEVS model would validate the DEVS Bridge model as well.

Part of the specification of a DEVS atomic model is the definition of the Time Advance function. The RISE server executes DEVS models using virtual time: the time of the simulation is advanced to the time of the next event to be executed. Because of this, all external events must be identified before the model executes, rather than being received as input from outside the executing model in real time. When a DEVS model is uploaded to the RISE server, all external events are captured in a file that specifies the event and the time at which it is input. The RISE server then executes the model, processing all events from the first external input event until there are no further external or internal events to be processed.

This lack of real-time interaction with the DEVS model is the reason why the DEVS Bridge treats the DEVS model as an engine for calculating a response to messages received from the rest of the synthetic environment. This is also why the state of the entities represented by the DEVS model is maintained by the DEVS Bridge and included in the model definition when it is sent to the RISE. A possible area of investigation would be to modify

the RISE server to support real-time DEVS, where the model is run in real time instead of virtual time. This would allow the DEVS model to execute as a proper federate in the larger federation, rather than requiring the DEVS Bridge to simulate time advance by running the entire model from start to finish with different input events.

6 Conclusion

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 organizations, 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 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 key entities in the C-BML and MSDL specifications were discussed. 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.

The main contribution of this work 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.

Another contribution of this research 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 research: formatted messages and state information specification.

One more contribution is the verification of the proposed architecture by implementing an application that 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.

By defining an architecture that adds a simulation interoperability interface to the RISE server, this research 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 floods or forest fires 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 behavior of people in the area, or supply information to a CGF system or 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.

Declaration of conflicting interest

The authors declare that there is no conflict of interest.

Funding

This research received no specific grant from any funding agency in the public, commercial or not-for-profit sectors.

References

1. Blais C, Gustavson P, Gustavsson PM, et al. An architecture for demonstrating the interplay of emerging SISO standards. In: *fall simulation interoperability workshop (FallSIW)*, Orlando, FL, 2006, http://www.sisostds.org/DigitalLibrary.aspx?EntryId=27053file06F-SIW-069.pdf.
2. Wainer G. *Discrete-event modeling and simulation: a practitioner's approach*. Boca Raton, FL: CRC Press, 2009.
3. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules, 1516–2010, 2010.
4. IEEE Standard for Distributed Interactive Simulation—Application protocols, 1278.1–2012, 2012.
5. MacQuarrie D, Taff C, Asselstine B, et al. Simulation-C2 interoperability through data mediation: the virtual command and control interface. In: *proceedings of the 2008 summer computer simulation conference*, Edinburgh, Society for Modeling & Simulation International. 2008.
6. Simulation Interoperability Standards Organization (SISO) guide for: Coalition Battle Management Language (C-BML) phase 1, SISO-GUIDE-002–2012-DRAFT, 4 April 2012.

7. Simulation Interoperability Standards Organization (SISO) guide for Base Object Model (BOM) use and implementation, SISO-STD-003.1–2006, 31 March 2006.
8. Simulation Interoperability Standards Organisation (SISO) standard for Military Scenario Definition Language (MSDL), SISO-STD-007–2008, 14 October 2008.
9. Al-Zoubi K and Wainer G. RISE: REST-ing heterogeneous simulations interoperability. In: *proceedings of the 2010 winter simulation conference*, Baltimore, MD, 2010.
10. Guidance, rationale, and interoperability modalities for the real-time platform reference Federation Object Model (RPR FOM), Version 1.0, 10 September 1999.
11. Multilateral Interoperability Programme (MIP) implementation guidance, MIG; Edition 3.1.10, 25 April 2013.
12. The Joint C3 Information Exchange Data Model, JC3IEDM Main, 14 February 2012.
13. Blais C and Alstad A. Coalition Battle Management Language (C-BML) phase I standard: trial use findings and next steps. In: *NPS papers and presentations for simulation interoperability standards organization's simulation interoperability workshop*, 2011, https://www.sisostds.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=34904&language=en-US&PortalId=0&TabId=105.
14. Bowers FA and Prochnow D. JTLS-JCATS federation support of emergency response training. In: *proceedings of the 2003 winter simulation conference*, New Orleans, LA, 7–10 December 2003, pp.1052–1060.
15. Cowdale A. Simulation modelling in support of emergency fire-fighting in Norfolk. In: *proceedings of the 2003 winter simulation conference*, New Orleans, LA, 7–10 December 2003, pp.1707–1710.
16. Brady TF. Emergency management: capability analysis of critical incident response. In: *proceedings of the 2003 winter simulation conference*, New Orleans, LA, 7–10 December 2003, pp.1863–1867.
17. Tao C, Hong-yong Y, Rui Y, et al. Integration of GIS and computational models for emergency management. In: *2008 international conference on intelligent computation technology and automation (ICICTA)*, Changsha, Hunan, 20–22 October 2008, pp.255–258.
18. Huang Y, Wang J, Jiang R, et al. Simulation and evaluation for the emergency management under the situation of fatal disaster. In: *2010 international conference on multimedia communications (Mediacom)*, Hong Kong, 7–8 August 2010, pp.258–262.
19. Aringhieri R. An integrated DE and AB simulation model for EMS management. In: *2010 IEEE workshop on health care management (WHCM)*, Venice, 18–20 February 2010, pp.1–6.
20. Rahmat MH, Annamalai M, Halim SA, et al. Agent-based modelling and simulation of emergency department re-triage. In: *IEEE business engineering and industrial applications colloquium (BEIAC)*, 7–9 April 2013, pp.219–224.
21. Sarjoughian HS and Zeigler BP. The role of collaborative DEVS Modeler in federation development. In: *proceedings of the 1999 fall simulation interoperability workshop*, Orlando, FL, 12–17 September 1999.
22. Mittal S, Risco JL and Zeigler BP. DEVS-based simulation web services for net-centric T&E. In: *summer computer simulation conference 2007 (SCSC 2007)*, San Diego, CA, 15–18 July, 2007.
23. Moallemi M, Wainer G, Jafer S, et al. Simulation of mobile networks using Discrete Event System specification theory. In: *proceedings of the 16th communications & networking symposium*, ACM, 7–10 April 2013, p.13.
24. Fielding R. *Architectural styles and the design of network-based software architectures*. PhD Dissertation, University of California, Irvine, CA, 2000.
25. Cappelaere PG, Frye SW and Mandl D. Flow-enablement of the NASA SensorWeb using RESTful (and secure) workflows. In: *proceedings of the 2009 IEEE aerospace conference*, Big Sky, MT, 7–14 March 2009, pp.1–7.
26. Rouached M, Baccar S and Abid M. 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.
27. Qian Z and Xianglong W. The research and implementation of a RESTful map mashup service. In: *2010 second international conference on communication systems, networks and applications (ICCSNA)*, 2010, pp.401–403.
28. Lanthaler M and Gutl C. 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 September 2011, pp.15–22.
29. Tosic M and Manic M. 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 November 2011, pp.38–43.
30. Pan B and Liang K. 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.
31. Bo C, Qiao X, Budan W, 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)*, Honolulu, HI, 24–29 June 2012, pp.620–622.
32. Al-Zoubi K and Wainer G. Using REST web-services architecture for distributed simulation. In: *ACM/IEEE/SCS 23rd workshop on principles of advanced and distributed simulation (PADS '09)*, 22–25 June 2009, pp.114–121.
33. Ribault J and Wainer G. Simulation processes in the cloud for emergency planning. In: *12th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid)*, Ottawa, 13–16 May 2012, pp.886–891.
34. Pullen JM, Corner D and Singapogu S. Scripted battle management language web service version 1.0 operation and mapping description language. In: *IEEE spring simulation interoperability workshop*, San Diego, CA, March 2009.
35. Turnitsa C. Extending the levels of conceptual interoperability model. In: *Proceedings of the IEEE Summer Computer Simulation Conference*, IEEE CS Press, 2005.
36. Ryan P and Oliver W. Representation of unmanned aerial systems in distributed simulation. In: *proceedings of the fall*

simulation interoperability workshop, Orlando, FL, 8–12 September 2014, pp.118–124.
37. Roos K, Solum E, Horn K, et al. Interoperability and harmonization analysis of the German Maritime Federation Object Model (GMF) in relation to SISO RPR FOM with possible extensions in a multinational forum. In: *proceedings of the fall simulation interoperability workshop*, Orlando, FL, 8–12 September 2014, pp.30–48.
38. Mifsud M, Morris N and Brook A Lessons learnt from delivering a readily-available, agile, distributed simulation capability to support research and development. In: *proceedings of the fall simulation interoperability workshop*, 8–12 September 2014, pp.125–131.
39. Military doctrine, http://en.wikipedia.org/wiki/Military_doctrine#Sources (accessed 18 March 2015).
40. Wombwell JA. *Army support during the Hurricane Katrina disaster*. Fort Leavenworth, KS: US Army Combined Arms Center, Combat Studies Institute Press, 2011.
41. Davidson A and Wainer G. Specifying truck movement in traffic models using Cell-DEVS. In: *proceedings of the 33rd annual simulation symposium (SS 2000)*, Washington, DC, 16–20 April 2000.
42. Wang S, Van Schyndel M, Wainer G, et al. DEVS-based building information modeling and simulation for emergency evacuation. In: *proceedings of winter simulation conference*, December 2012, pp.1–12.
43. Liu Q and Wainer G. Accelerating large-scale DEVS-based simulation on the cell processor. In: *proceedings of the 2010 spring simulation multiconference*, San Diego, CA, 2010.
44. Risco-Martin JL, de la Cruz J, Mittal S, et al. eUDEVS: executable UML with DEVS theory of modeling and simulation. *Simulation* 2009; 85: 750–777.
45. Kapos, G, Dalakas, V, An integrated framework for automated simulation of SysML models using DEVS. *Simulation* 2014; 90: 717–744.
46. Gautreau B, Khimeche L, Remmersmann T, et al. Lessons learned from NMSG-085 CIG Land Operation demonstration. In: *spring simulation interoperability workshop (2013 Spring SIW)*, San Diego, CA, 8–12 April 2013, pp.176–185.
47. Standardization for C2-Simulation Interoperation (MSG-085) web site, http://www.cso.nato.int/ACTIVITY_META.asp?ACT=1957 (accessed September 2013).

Author biographies

Elizabeth Hosang obtained a BSc in computer engineering at the University of Manitoba (1992) and a MSc in electrical and computer engineering at Carleton University (2014). She is a software architect with CAE Defense & Security, Canada Region, where she works on data fusion and army C2 systems.

Gabriel A Wainer (SMIEEE, SMSCS) received a PhD (1998, with highest honors) at the University of Marseille, France. In July 2000, he joined the Department of Systems and Computer Engineering at Carleton University, where he is a full professor. He has authored three books and over 300 research articles; he has edited four other books and helped organize a large number of conferences, being one of the founders of SimAUD, SIMUTools and TMS-DEVS. He was vice-president conferences and vice-president publications of the SCS (Society for Modeling and Simulation International), where he is a current member of the board of directors. He is the special issues editor of *SIMULATION*, member of the editorial board *of IEEE Computational Science and Engineering, Wireless Networks* (Elsevier) and the *Journal of Defense Modeling and Simulation* (SCS). He received the IBM Eclipse Innovation Award, the SCS Leadership Award and various best paper awards; also, the first Bernard P. Zeigler DEVS Modeling and Simulation Award (2010), the SCS Outstanding Professional Award (2011), Carleton University's Mentorship Award, the SCS Distinguished Professional Achievement Award (2013) and Carleton University's Research Achievement Award (2005 and 2014).