Interfacing DEVS and Visualization Models for Emergency Management

Mohammad Moallemi, Shafagh Jafer, Ahmed Sayed Ahmed, Gabriel Wainer
Dept. of Systems and Computer Engineering
Carleton University, Centre of Visualization and Simulation (V-Sim)
{moallemi, sjafer, asahmed ,gwainer}@sce.carleton.ca

Keywords: Cell-DEVS, Emergency simulation, real-time DEVS.

Abstract

We introduce a method to integrate Cell-DEVS models with DEVS-based robotic agents and an advanced Immersive environment for Emergency Management. The emergency is handled by an autonomous robot controlled by a real-time DEVS model. The model controlling the robot interacts with a simulation for emergencies, receiving real-time data about its location on a cell space. The immersive environment is used to visualize the emergency and its management. The simulation results of both the cell-DEVS emergency model and the DEVS-based robotic first responder are visualized dynamically at real-time. The goal is to show how to integrate cellular modeling in a real-time platform and the DEVS formal framework as a collaboration mechanism. The real-time visualization allows for supervisory control of the emergency and first responders activities.

1. INTRODUCTION

Emergency simulation has received increasing attention in recent years and several research works have been developed and proposed for this purpose. Specifically, disaster management and evacuation strategies are the two most important subjects within this field. Due to the devastating occurrence and destructive impacts of emergencies, it is impractical to perform real-world studies of this nature. An alternative to field based experiments is modeling and simulation (M&S) [1]. Generally, such simulations are large-scale programs, which in return raise the need for efficient simulation engines. Modeling, simulation, and visualization techniques can help address many of the challenges in emergency response planning [1].

Since emergency crisis are processes that distribute over both time and space, simulations should take into consideration the system evolution in both time and space. In this paper we present an integrated emergency management system based on the Discrete-Event System Specification (DEVS) [3] and Cell-DEVS [4]. We focus on the interoperation of different DEVS and Cell-DEVS models with the purpose of integrating emergency simulation with emergency management. The emergency simulation is based on Cell-DEVS, and the emergency management is performed by a robotic agent controlled by a DEVS model to respond to the emergency at real-time (RT) using the CD++ toolkit [5][6]. We also use a visualization engine that takes the results of the emergency simulation and the emergency management as input and produces 3-D visualizations of the simulation scenarios.

2. RELATED WORK AND MOTIVATION

A number of M&S applications exist for studying individual aspects of emergency response scenarios. However, a number of simulation tools have to be integrated to address multiple aspects of a single disaster event. In [7], a framework for M&S for emergency response is presented which systematically integrates M&S tools to address the overall response. In [8], the authors integrate gaming and simulation systems for training decision makers and responders to work together as a team. A discrete-event environment introduced in [9] presents the processes of analysis and design of a multi-agent system for a crisis response organization with the purpose of building a simulation testbed to experiment with different coordination mechanisms. Also, virtual simulation with RT emergency response has become an emerging technique that plays a key role for efficient emergency management systems due to its capabilities to provide RT system observations [10][11][12].

We introduce a simulation-driven architecture for integrating emergency simulation with robotic first responders moving towards the emergency locations, which are spread out on the field. The robot is placed on a grid corresponding to the simulated emergency area, and reaches every location, dealing with the emergency. Our work differs from the previously mentioned works in three different ways:

1. The RT cellular emergency simulation is an on-demand data source of the scenario, which is to be used by the robot. A supervisory control station can be used to update the emergency simulation data with a real emergency situation and the information of the area.
2. This multimodel combines a Cell-DEVS cellular model, a DEVS-based robotic controller, and virtual reality visualization. This component-oriented approach provides model reusability and interoperability, allowing integration or replacement of any of the components.
3. Using a simulation-driven approach for controlling the robot allows testing the robot controller in a fully simulated environment, then using the same model for controlling a real robot. Model-continuity from early simu-

lation stages to finally embedding it on the hardware speeds up the development process while increasing the reliability of the product and reducing risk and cost.

3. THE SYSTEM ARCHITECTURE

Figure 2 presents a detailed overview of the system. The emergency simulation sub-system is in charge of the Cell-DEVS emergency model. It communicates with the DEVS emergency response sub-system informing it about the dimensions of the emergency area, and sends updates about the location of the emergency team on the grid. At the same time, it also sends this information to the visualization sub-system providing it with RT data about the scene. The DEVS-based control model uses the emergency information received from the Cell-DEVS engine to respond to the emergency. Based on these commands, the robot moves on the simulated grid area and deals with the emergencies one after another. The emergency response sub-system dynamically updates the emergency simulation and visualization sub-systems when emergencies have been solved. This process continues until all emergencies are dealt with. The visualization sub-system produces 3D scenes from the dynamically received updates from both the emergency simulation and the emergency response sub-systems.

3.1. Emergency Simulation

We used a Cell-DEVS model to represent an area that has a number of emergency locations (e.g. road bombs or explosions) which are ignited randomly during the simulation. The model is defined according to the conventions of Cell-DEVS using CD++ toolkit. It consists of an atomic component tested with various dimensions from 30 x 30 (900 cells) to 200 x 200 (40,000 cells). Initially, the model will locate different emergency locations on the plane, and it will locate the current position of the emergency team in the field.

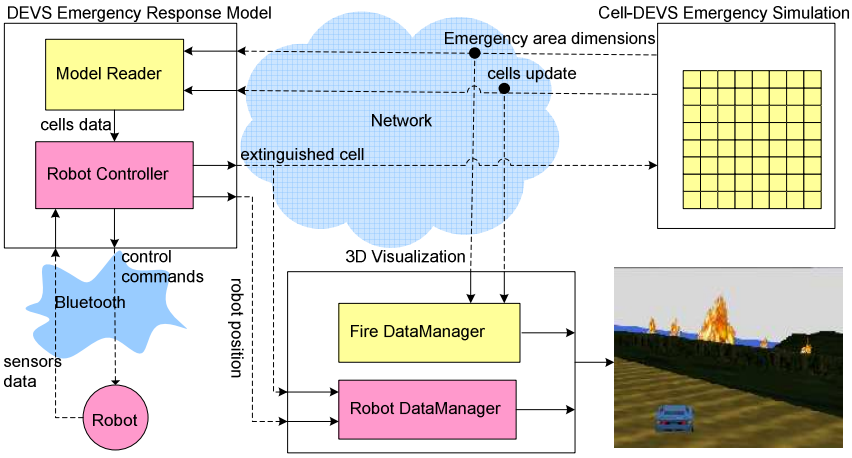

Figure 1. System Architecture

In order to run the model in RT, we have modified the CD++ simulation engine, making the virtual time-advance of the DEVS simulation algorithm replaced by a RT version of the algorithm. This allows the emergency simulation to interact with the emergency response and the visualization sub-systems in RT. We have also added a generic interface to the simulation engine enabling it to interact with the external environment (e.g., an external network). The simulator sends the dimensions of the cell space at the start of the simulation, submits any cell updates, and at the same time receives input to the cellular model using the message structure (Section 6).

4. ROBOTIC FIRST RESPONDER AGENT

In this section, we discuss the details of the design and implementation of a DEVS-based controller for the autonomous first responder robot. The controller model can be also used for swarm robotic applications [13] in which a large number of homogeneous autonomous robots are engaged in an activity. We used the e-puck robotic kit [14] as a prototype for first responder robot to deploy the simulation-driven controller developed for emergency response system. The E-puck is a small mobile robot capable of moving and spinning and is equipped with sensors and motors. It uses eight infrared distance proximity sensors (IR) to detect obstacles around it. There are eight LEDs mounted on the top shaping a ring. The robot can interact with a PC via Bluetooth connection. The reason, why we use a DEVS model for the robotic agent and not doing this in the Cell-DEVS model, is the limitations of a cellular modeling formalism.

4.1. The Logical Controller

Initially, the robot model receives the size of the cell space and builds a copy of the cellular space for itself. The robot also receives the updates of cell values from the cell-DEVS model and marks the changes on its own copy.

The controller model consists of two levels of controls: a High-level and a Low-level control. The first on is responsible for path planning towards the closest emergency location (using the data provided by the cell space); the low-level control is responsible for avoiding obstacles.

a) Higher Level Controller (HLC)

This controller must locate the emergency locations on the cell space, find the closest one and steer the robot towards it. The HLC must know the current position of the robot on the cellular space. Afterwards, it calculates the distance to each location and chooses the closest one as the target emergency site to arrive at. By knowing the length of each cell (which is mapped to an area on the ground) and the speed of the robot, we update the position of the robot. The Cell-DEVS model uses the 9 nearest neighbors and there are eight regions defined in the space (Figure 2.a). The HLC decides the next movement based on the region where the incident is located in. This process continues until the robot arrives at the target emergency location. Figure 2.b shows a sample scenario where initially, the robot locates the emergency in R2 and moves one step to North east. Again the target is in R2, and also in the next step. Therefore after moving three steps to North East, the robot finds the fire in R1, where the robot catches it by moving one step to the North.

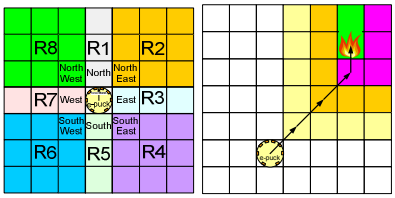

Figure 2. a) Region definitions b) HLC example

b) Lower Level Controller (LLC)

In outdoor emergency control missions, first responders usually work in hazardous environments. The robot must thus avoid obstacles: large obstacles that are marked on the cell space (e.g. rivers, houses, cliffs) and smaller obstacles, detected by the robot sensors (e.g. trees, stones). The LLC is applied after the HLC, deciding based on the neighborhood information acquired from the local cell space and the sensor inputs. In other words, HLC has a top view to the entire cell space while LLC only observes the neighborhood and the sensor range area surrounding the robot. If LLC detects obstacles in the direction determined by HLC, a new open direction adjacent to the previous direction is assigned to the robot. The deviation from the original path determined by HLC is restored in the next steps by HLC. Figure 3 illustrates an example scenario in which the robot tries to reach the emergency with a river and the stone in the path. The HLC steers the robot towards the emergency until the robot reaches the river (which is marked in the cell space, thus the LLC guides the robot towards the NW). Then, the LLC corrects the HLC decisions until the robot reaches the bridge. The red arrows show the original directions determined by HLC, which were corrected later by LLC.

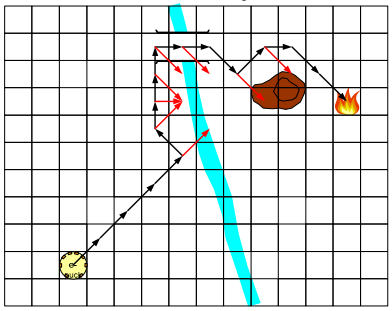

Figure 3. Sample scenario for HLC and LLC

4.2. Controller Model Specifications

We divided the model responsibilities into two parts. The *Model Reader* is responsible for creating the local cell space, updating the cell space by receiving the updates from the Cell-DEVS engine, and signaling the *Controller* component to make path-planning decisions. The *Controller* implements the HLC and LLC algorithms, sending control commands to the robot and informing the visualization engine about the robot movements.

Figure 4 depicts an abstract representation of the behavior of the two components using a DEVS graph [15] that summarizes the behavior of the DEVS atomic component by rendering the states, transitions, inputs, outputs and state durations of the atomic component. The *Model Reader* starts in the state *wait for dimension*, where it is waiting to receive the dimensions of the cell space from the Cell-DEVS engine. As soon this happens, it builds a local copy of the cell space, and then changes to *idle* (which corresponds to the movement period of the robot). During the *idle* state, the *Model Reader* also receives cell space updates and marks them on the local copy. If an emergency update is received, it is added to the emergency list. At the end of this state, the *Controller* is signaled to carry out the next movement.

The *Controller* starts by sending the initial position of the robot to the visualization engine and *stop*s (a state where it receives periodic signals from the *Model Reader*). If there is an emergency location in the emergency list, the *Controller* changes to *Calculate next step* and the following tasks are executed in the corresponding external transition: sort the emergency list, find the closest site, apply the HLC and LLC algorithms, and calculate the next step. Based on the result of the control algorithms, the *Controller* changes to one of the movement states, the output function triggers the movement commands for the robot, and the next step information for the visualization engine is sent. This sequence is repeated until it reaches the emergency site. In that case, it changes to *prepare extinguish*. After this, it outputs the stop command to the robot, informs the Cell-DEVS and visualization engines about the emergency restraint, and transitions to the *stop* state, where it waits for the next location.

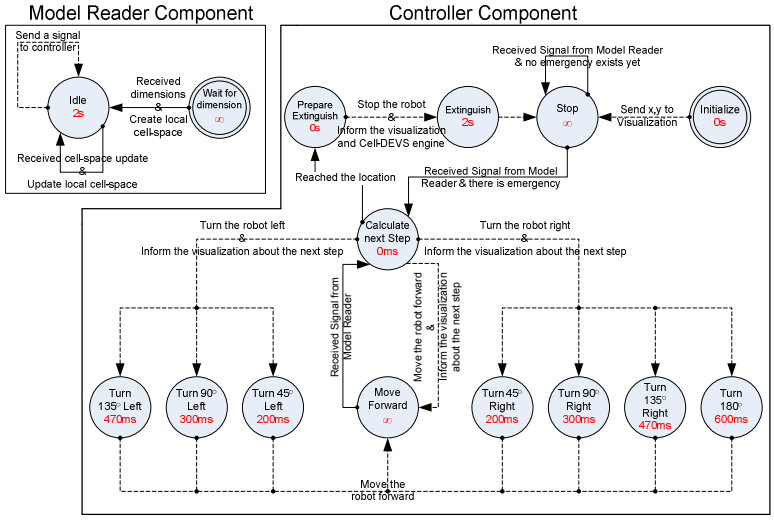

Figure 4. DEVS Graph of the robot controller

5. CONTROLLER IMPLEMENTATION

E-CD++ is an open-source embedded RT DEVS-based modeling, simulation and application development environment [16], built as a Real-Time extension of the CD++ simulator. ECD++ allows defining driver interface functions for each input and output port of a DEVS model, in which the integer I/O values of a DEVS system are translated to signals to the external environment.

Figure 5 shows a screenshot of the different stages of a sample scenario in which the robot starts on cell (2,1) and moves to the closest emergency on the first row. The cellular animation tool renders a basic animation of the cellular model by reading the log file of CD++ simulator. In the end, the robot arrives at two other emergency locations that are the closer to the previous ones.

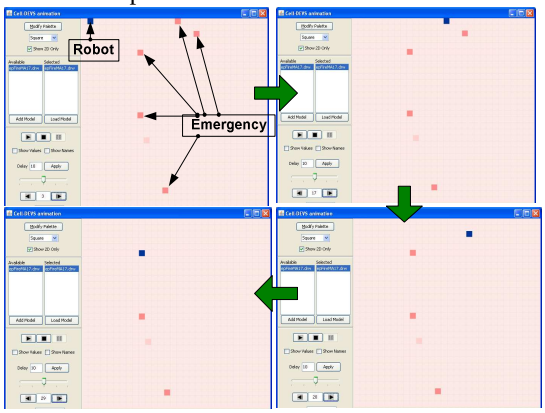

Figure 5. Animation of the emergency management

6. VISUALIZATION

Visualization of emergency behavior can provide a number of benefits. First, it provides an interactive environment to verify the accuracy of these models by comparing the results of an actual emergency with the output of a simulated version. Once the model is validated, it can then be used to predict not only the behavior of an existing emergency, but also the consequences of preventive measures. Displaying these predictions in a visually informative manner allows the Emergency Department to better educate the first responders on existing emergency hazards. Furthermore, enabling interactive manipulation of the simulation along with the visualization allows for emergency training in terms of resource allocation and emergency behavior. While experimentation would be risky and costly to perform in a real-life situation, these risks can be mitigated by simulating untested approaches first. 3D user interfaces provide a more intuitive form of interaction. Additionally, high-fidelity graphics enables an observer in better comparing a simulated emergency to a historic emergency.

6.1. 3D Visualization Engine Description

The 3D visualization engine is used to visualize the simulation output results of both the emergency simulation model and the robotic first responder agent. The visualization engine is implemented using Vega Prime and OpenGL [14]. Vega Prime is a high-performance software environment for RT simulation and virtual reality applications. It serves as an application programmer interface (API) consisting of a graphical user interface called LynX Prime and Vega Prime libraries and header files of C++-callable functions. 3D scenes are rendered using 3D *openflight* models. The terrain model consists of trees, different buildings, roads, etc. the robotic agent is represented by a 3D truck model. We can control how the effect of environment and time of the day in the 3D scene visualization. Figure 6 show in a window that is divided into two channels; one for perspective view of 3D scene (on the left), and the other chan-

nel is for orthographical view of the 3D scene which acts as 2D Map of the area (on the right).

- On the perspective view of the first channel, the movement of the emergency responder truck is displayed as a 3D model representing the robot, and it is observed using a fixed camera. The observer view can be changed to five positions (back, front, left side, right side, or rotate) around the emergency responder.
- On the orthographical view of the second channel, a red grid represents the cellular grid of the emergency simulated area. The emergency locations received from the Cell-DEVS engine are rendered by flashing yellow circles and the emergency responder truck is represented by a white circle. The white circle changes to green when the robot extinguishes an emergency location in the scene, after which the emergency special effects and the flashing yellow circle is removed from the 3D scene. The orthographical view is capable of zooming in and out and the cellular grid can be removed for a better view (see Figure 6).

Figure 6- 3D Visualization Engine zoomed map

6.2. 3D Visualization Engine Implementation

The 3D visualization subsystem (implemented in Visual C++) is shown in Figure 7. It consists of two main components: *(i)* the *Receiver*, which receives the data from the DEVS-based robot model and the Cell-DEVS emergency model, *(ii)* the *Visualizer*, which is responsible for the display of the visualization scene.

The *Receiver* component is a separate thread, spawned for receiving the emergency and suppression data.

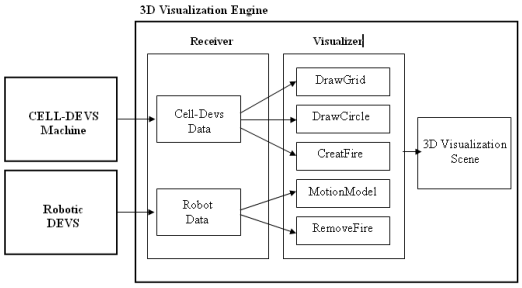

Figure 7- 3D Visualization Engine Architecture

The 3D visualization engine is capable of deploying different 3D terrain *openflight* models and different cellular areas (dimensions and initial values) without changing the code of the visualization.

6.3. Global Message Structure

The collaboration of the three components is based on a global message structure transferred over a network infrastructure. The *network_struct* contains the following five data fields:

1. *msg_id*: an integer data type used to decode the type of the message and the value of the next fields in the message. There are generally five types of messages:
   - The *dimension message* carries the size of the cell space from the Cell-DEVS engine to the DEVS and visualization at the start of the execution.
   - The *robot initial location message* carries the initial coordinates of the robot from the DEVS engine to the visualization.
   - The *cell space update message* carries the cell value changes during the execution from the Cell-DEVS engine to the DEVS and visualization.
   - The *next movement message* carries the direction of the next movement at the start of each step from DEVS to the visualization engine.
   - The *extinguish message* carries the location of the emergency that has been extinguished by the robot, from the DEVS sub-system to the Cell-DEVS and visualization sub-systems.
2. *x*: used to carry the horizontal axis value (the horizontal dimension or the horizontal coordinate).
3. *y*: used to carry the vertical axis value (the vertical dimension or the vertical coordinate).
4. *dir*: carries the next direction. Directions are the same as the regions shown in Figure 2.a.
5. *value*: carries the value of the cell and is used in the cell update message.

7. CONCLUSIONS

We presented a DEVS-based emergency management simulation and visualization system. Our system offers a robust software framework to make the RT emergency response system more flexible and more scalable. A robotic agent acting as a first responder is placed in a virtual environment generated from a Cell-DEVS emergency simulation. The controller of the robot is a DEVS-based emergency response model that interacts with the emergency simulation through messaging and is informed about the map of the area and the location of the incident (e.g. road bombs, fire, explosions, etc). Both the emergency simulation and the emergency response sub-systems run in RT and communicate with each other and with a 3D visualization engine. The purpose of the visualization system is to provide 3D scenes and to visually monitor the activities of the robotic first responder. Although the emergency model is a simulation, it can be simply replaced with more complex emergency simulation models or a real emergency database fed from real-world data.

The proposed generic interface and message structure that enables the emergency simulation, the emergency response, and visualization sub-systems interchange data, enables our system to simulate emergency management in RT under various conditions. One of the future extension plans of this work is the integration of Google-map™ free virtual reality web service with visualization engine and inject the terrain data to the Cell-DEVS engine. Some videos of this work can be found in [17].

8. REFERENCES

[1] Kincaid, J. P., Donovan, J., Pettitt, B., "Simulation techniques for training emergency response". International Journal of Emergency Management, vol. 1, pp. 238-246, 2003.

[2] Jain, S. C.R. McLean, "Modeling and Simulation of Emergency Response: Workshop Report, Relevant Standards and Tools," National Institute of Standards and Technology Internal Report, NISTIR-7071. 2003.

[3] Zeigler, B. Praehofer, H. Kim, T. 2000. Theory of Modeling and Simulation, 2nd ed. Academic Press.

[4] Wainer, G., "Applying Cell-DEVS Methodology for Modeling the Environment". In *Simulation, Transactions of the SCS*. Vol. 82, No. 10, 635-660. October 2006.

[5] Wainer, G., "CD++: A toolkit to develop DEVS models". Software–Practice and Experience, 32:1261-1306. 2002.

[6] Wainer, G., "Discrete-event modeling and simulation; a practitioner's approach". CRC/Taylor & Francis. 2009.

[7] Jain, S., McLean, C., "A Framework for Modeling and Simulation For Emergency Response". Proceedings of the Winter Simulation Conference. 2003.

[8] Jain, S., McLean, C., "Integrated simulation and gaming architecture for incident management training". Proceedings of 37th Winter simulation conference, Orlando, FL. 2005.

[9] Gonzalez, R., "Analysis and design of a multi-agent system for simulating a crisis response organization". Proceedings of the International Workshop on Enterprises & Organizational Modeling and Simulation, Amsterdam, The Netherlands. 2009.

[10] Boukerche, A., Zhang, M., Pazzi, R., "An adaptive virtual simulation and RT emergency response system". Proceedings of the IEEE international conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems, Hong Kong, China. 2009.

[11] McGrath, D., Hunt, A., Bates, M., "A Simple Distributed Simulation Architecture for Emergency Response Exercises". Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications, p.221-228, October 10-11, 2005.

[12] Liu, K., Shen, X., Georganas, N., El Saddik, A., Boukerche, A., "SimSITE: The HLA/RTI Based Emergency Preparedness and Response Training Simulation". Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2007.

[13] Sahin, E., Spears, "Swarm Robotics: State-of-the-art Survey". Lecture Notes in Computer Science 3342, Springer-Verlag, 126–14. 2005.

[14] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. "The e-puck, a robot designed for education in engineering", In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, volume 1, pages 59–65, 2009.

[15] Zeigler, B.P., Song, H., Kim, T., Praehofer, H., "DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems". Proceedings of HSAC, LNCS, Vol. 999. Ithaca, NY. 1995.

[16] Moallemi, M., Wainer, G. A., "Designing an Interface for Real-Time and Embedded DEVS". Spring Simulation Conference, DEVS Symposium, Orlando, Florida, USA, 2010.

[17] Emergency Management 1. Available via: <http://www.youtube.com/watch?v=9aNVZRkrtC8>. [Accessed December, 2010].