VCELL: A 3D Real-time Visual Simulation in Support of Combat

Ahmed Sayed Ahmed, Mohammad Moallemi, Gabriel Wainer, and Samy Mahmoud
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, Ontario, Canada K1S 5B6
{asahmed, moallemi, gwainer, and mahmoud}@sce.carleton.ca

Keywords: Cell-DEVS, ABD, CAS, 3D Real-time Visual Simulation, Collaborative Modeling

Abstract

We present the development of a 3D real-time visual Cellular Agent model (VCELL). VCELL is used for simulating land combat and is collaboratively modeled using a cellular agent model based on the Cell-DEVS formalism and an advanced immersive environment based on a 3D real-time visual simulation. VCELL is used to enhance and improve the random selection caused by movement algorithms of Agent-based distillation (ABD). The model includes a highly modular collection of software packages designed to facilitate the development of device-independent simulation for land combat. The immersive environment is used to visualize the land combat. The simulation results of the Cell-DEVS agent model are visualized dynamically in real-time. The goal is to show how to integrate cellular modeling in a real-time platform and 3D real-time visualization as a collaboration mechanism to enhance movement algorithms in land combat. The 3D real-time visualization allows for supervisory control of the land combat activities.

1. INTRODUCTION

Land combat warfare has received increasing attention in recent years and several research works have been developed and proposed for this purpose. It has been shown that warfare can be described by nonlinear behaviors [1–3]. Combat model can be presented as a complex adaptive system (CAS) [2], where the global behavior depends on local interaction of agents. Complex adaptive systems can be considered as a special case of complex system [4–7] i.e, a dynamical system composed of many nonlinearly interacting agents. Agent-based simulations can be deployed into military operations, which are usually known as Agent-based distillation (ABD) or agent-based simulation. These agent-based distillations (ABD) are low-resolution abstract models, used to explore different aspects of land combat operations that will help decision makers to quickly investigate different scenarios in a real battle condition. An agent-based simulation is designed by simple behavioral rules. A number of agent-based simulation applications for land combat are surveyed in [1–3, 8–14]. Agent-based simulation toolkits use a function for the agent's movement in the combat battlefield space based on their movement algorithms.

Since land combat movement algorithms can be distributed over both time and space, land combat simulations should take into consideration the system evolution in both time and space. In this paper, we present a collaborative land combat model based on the Cell-DEVS formalism [15] and 3D real-time visualization to develop new classes of land combat movement algorithms.

We present a hybrid model to solve the randomization problem caused by a random moving selection between the cells in tie, which is produced by the movement algorithms of most ABD toolkits. We propose to solve this problem by collaboration between an agent based on Cell-DEVS formalism [15] and a visual Agent simulation based on a 3D real-time visualization simulation in real-time. The visual agent simulation allows us to visualize the land combat simulation scenarios in a 3D scene. We also propose to reduce the programming time consumed to develop or modify the scenario tactics for combat in real-time visual simulation by using Cell-DEVS, which allows us to use simple rules to write the different scenarios.

2. RELATED WORK

Various development efforts for combat have been designed based on agent-based simulation. One of such efforts is the Irreducible Semi-Autonomous Adaptive Combat (ISAAC) [1, 2], and its extension the Enhanced ISAAC Neural Simulation Toolkit (EINSTein) [1, 8] designed on the US Marine Corps Combat Development Command. BactoWars was developed by Land Operations [9]. The Map Aware Non-uniform Automata (MANA) [3, 10] was provided by New Zealandś Defence Technology Agency. The Conceptual Research Oriented Combat Agent Distillation Implemented in the Littoral Environment (CROCADILE) [11] and the Warfare Intelligent System for Dynamic Optimization of Missions (WISDOM) [12–14] developed at the University of New South Wales at the Australian Defence Force Academy. All of these development tools use a function for the agent's movement in the space. A research introduced in [16] ex-

amined the movement algorithms of MANA . Movement algorithm of agents within the EINSTein and MANA ABDs was modified by Grieger [16]. In such combat simulations, an agent always moves to the cell with the maximum weight. If a tie happens, the agent selects randomly between the cells in tie. Due to this randomization, the stability of the solution may be affected and the outputs of this combat simulation are not guaranteed [17].

Our work presents a collaborative 3D real-time visual Cellular Agent model (VCELL) and a Cellular Agent simulation in real-time. The agents are divided into two teams; the blue team in the 3D visualization agent sub-model, and red team in the Cell-DEVS agent sub-model. These sub-models collaborate via a network connection. Our work differs from previous researches as follows:

- This work incorporates two components: a Cell-DEVS agent simulation and a 3D visualization agent simulation. This component-oriented approach provides model reusability and interoperability, allowing for integration or replacement of any of the two components.
- It uses a VCELL model for removing the random movement problem in the blue team providing 3D visualization agent that gets the real position of agents in the combat. This guarantees the combat simulation output for the blue team.
- The 3D visualization agent simulation is an on-demand data source for the combat scenario. Real fighters can be invoked in the 3D visualization agent simulation to update the agent simulation data with a real situation.

2.1. Agent-Based Distillation (ABD)

The Lanchester Equations were considered to model and hypothesize combat attrition by defense analysts [1, 2]. The Lanchester Equations were presented by Lanchester in 1916 [18] as a set of linear dynamic equations that address attrition as a continuous function over time. In Lanchester Equations, Combat is modeled as a deterministic process that need an attrition-rate coefficient. Lanchester equations are easy to apply. Models based on mathematical equations and physical description of combat can only provide an ideal model of military operations that is too abstract and far from realistic. The shortcomings of the Lanchester equations have been listed and analyzed in [1, 3, 8, 11]. Research results in [1–3] shows that warfare can be considered as nonlinear behavior. Combat is considered as a complex adaptive system (CAS) [2]. A Multi-agent system (MAS) is a platform for studying CAS. The combatants are modeled as agents, usually with a set of pre-defined characteristics. These agents adapt, evolve and co-evolve with their environment [3,19]. This view of combat allows researchers to use agent-based simulations on military operations. The field is usually known as ABD or agent-based simulation (ABS). ABD emphasizes the concept of incorporation of agents in the environment [20]. Defense analysts have studied different behaviors of warfare based on ABD. Simulation is used to study and analyze the dynamics and behaviors of the system, which provide defense analysts with a useful tool for helping them in making decisions.

By modeling an individual constituent of a CAS as an agent, we are able to simulate a real world system by an artificial world populated by interacting processes. This is particularly effective to represent real world systems that are composed of a number of nonlinear interacting parts with a large space of complex decisions and/or behaviors to choose from such as those situations in real combat [2].

2.2. Cell-DEVS

Among the existing simulation techniques, DEVS (Discrete Event System Specification) formalism [21] provides a discrete-event M&S approach which allows construction of hierarchical models in a modular manner. DEVS is an increasingly accepted framework for understanding and supporting the activities of modeling and simulation. A real system modeled with DEVS is described as a composite of submodels, each of them being behavioral (atomic) or structural (coupled).

A cellular automaton is a discrete model which is composed of a network of cells that each cell has a finite number of states [22]. The state of each of the cells in time t is a function of states of its predefined neighbor cells in time t-1. Cell-DEVS [15] has extended DEVS, allowing the implementation of cellular models with explicit timing delays. A Cell-DEVS model is a lattice of cells, where each cell is a DEVS atomic component, holding state variables and a computing apparatus, which is in charge of updating the cell state according to a local rule-base. This is done using the current cell state and those of a finite set of nearby cells (called its neighborhood). Cell-DEVS improves execution performance of cellular models by using a discrete-event approach. It also enhances the cell's timing definition by making it more expressive. Each cell is defined as a DEVS atomic component, and it can be later integrated to a coupled component representing the cell space. Cell-DEVS atomic components are informally defined as in Figure 1.

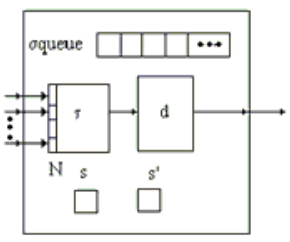

Figure 1. Description of a Cell-DEVS atomic component

Each cell uses N inputs to compute its next state. These inputs, which are received through the model's interface, activate a local computing function (τ). A delay (d) can be associated with each cell. The state (s) changes can be transmitted to other models, but only after the consumption of this delay. Once the cell behavior is defined, a coupled Cell-DEVS can be created by putting together a number of cells interconnected by a neighborhood relationship.

A Cell-DEVS coupled model is informally presented in Figure 2. A coupled Cell-DEVS is composed of an array of atomic cells, with given size and dimensions. Each cell is connected to its neighborhood through standard DEVS input/output ports. Border cells have a different behavior due to their particular locations, which result in a non-uniform neighborhood.

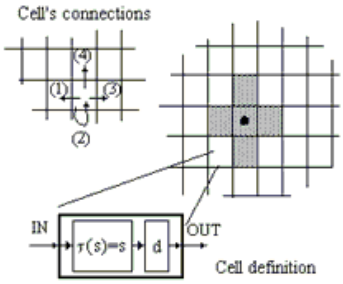

Figure 2. Description of a Cell-DEVS coupled component

CD++ [23] is a modeling software tool to implement the DEVS and Cell-DEVS models. DEVS Atomic models can be programmed and incorporated into a class hierarchy programmed in C++. Coupled models can be defined using a built-in specification language. Cellular models are built following the formal specifications of the Cell-DEVS formalism, and a built-in language is provided to describe declare the model and the associated rules. CD++ is built as a class hierarchy of models, where the user-defined model classes are united with the main simulation core classes to make a full hierarchy of simulation entities. CD++ includes an interpreter for Cell-DEVS models, which allows for automatic generation of atomic cells and allows for definition of the cell state change rules. The language is based on Cell-DEVS formalism, in which the model specifications include the size and dimension of the cell space, the shape of the neighborhood and borders can be defined. The cell's local computing function is defined using a set of rules with the form POSTCONDITION DELAY PRECONDITION. This syntax indicates that, when the PRECONDITION is satisfied, the state of the cell changes to the designated POSTCONDITION, whose computed value will be transmitted to other components after consuming the DELAY. If the precondition is false, the next rule in the list is evaluated until a rule is satisfied or there are no more rules.

2.3. 3D Visual Simulation

The rapid expanding technology in computer processing, storage, communications, and display capability has resulted in a rapid growth of software modeling and visual simulation [24]. Interactive 3D simulation is used in combat simulation, where there is a necessity during peacetime to train soldiers in order to gain successful results in wartime missions [25]. Training simulation systems can be treated as games, hence a large effort has been devoted recently, on learning and skill improvement of the trainees [26]. Digital Game-Based Learning [27] -as today and future learning style- is deployed as a method of learning and training, because it is motivating, and it is effective when used in a correct way

3. THE VISUAL CELL-DEVS AGENT (VCELL) ARCHITECTURE

The Visual CELL-DEVS Agent (VCELL), as shown in Figure 3, is composed of two main subsystems:

1. The CellAgent sub-model which is implemented using a Cell-DEVS model running in real-time.
2. The 3D real-time visual simulation (RTV)sub-model which is implemented using Vega Prime and OpenGL [28]

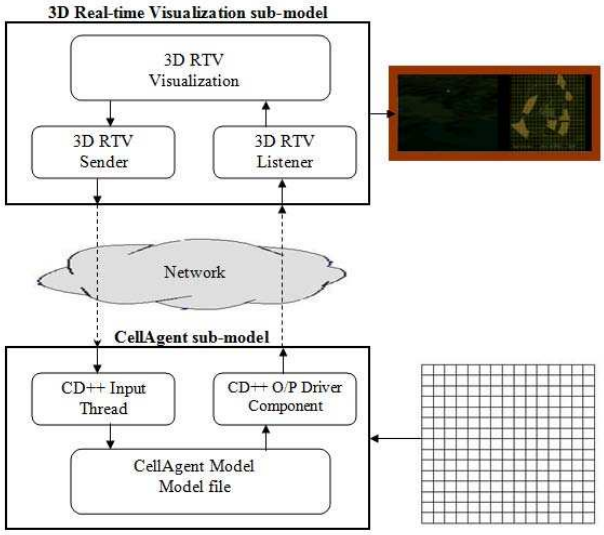

Figure 3. The Visual CELL-DEVS Agent (VCELL) Architecture

Each sub-model runs on a different machine, communicating using messages sent over a network. The CellAgent sub-model and the 3D real-time visualization sub-model run in real-time and they communicate via messages transferred through a network infrastructure. VCELL is a multi-agent simulation combat system which facilitates the analysis and understanding of land combat, and 3D real time visualization

simulation for tactics in land combat. By using VCELL, not only the analysts can understand the overall shape and dynamics of a battle and know the output of an operation but also combatant could be trained in the 3D visual real time simulation system. An agent in VCELL is characterized by some properties such as: capabilities, movements, communications and health. Agents can communicate by exchanging messages. The health can be defined as the level of energy for an agent. The level of energy of agents are defined by users. When an agent is attacked by the opponent agent type, its health depends on the number of the neighborhood agents and their health strength. Users can import different 3D terrain types in the real time visual model. The type of terrain affects the agent's movements.

The agent's movement depends on five different weights; agent healthy friend, agent injured friend, agent healthy opponent, agent injured opponent, and the flag. The movement is calculated at each simulation time step for each agent. The agent can move to another cell or decide to stay at the same cell. Each cell in the space cannot be occupied by more than one agent at a time. The decision making used by each agent to decide the direction to move, depends on the agent's personalities in the movement algorithm. The movement algorithms used in VCELL is the same than in EINSTein and MANA, but VCELL is not restricted to these algorithms only as we can apply different movement algorithms. The Movement Algorithm of EINSTein uses equation (1) to compute the penalty for the next location [1, 8]:

$$Z_{new} = \left(\frac{W_E}{E * R_S \sqrt{2}} \sum_{i=1}^{E} D_{i,new} \right) + W_F \left(\frac{D_{F,new}}{D_{F,old}} \right) \quad (1)$$

where

R_S Sensor range of agent about to move;
E Number of enemy entities within sensor range;
W_E Weighting towards enemy agents;
$D_{i,new}$ Distance to the ith enemy from the new location;
W_F Weighting towards the flag;
$D_{F,new}$ Distance to the flag from the new location;
$D_{F,old}$ Distance to the flag from the current location.

The Movement Algorithm of MANA uses equation (2) to compute the penalty for the next location [3, 10]:

$$Z_{new} = \left(\frac{W_E}{100 * E} \right) \left(\sum_{i=1}^{E} \frac{D_{i,new} + (100 - D_{i,old})}{100} \right) + \left(\frac{W_F}{100} \right) \left(\frac{D_{F,new} + (100 - D_{F,old})}{100} \right) \quad (2)$$

where

E Number of enemy entities within sensor range;
W_E Weighting towards enemy agents;
$D_{i,new}$ Distance to the ith enemy from the new location;
$D_{i,old}$ Distance to the ith enemy from the current location
W_F Weighting towards the flag;
$D_{F,new}$ Distance to the flag from the new location;
$D_{F,old}$ Distance to the flag from the current location.

Agents are encouraged to move closer to the opponent agent. The agent will always move to the cell with maximum weight. There is no tie in the real-time simulation sub-model, but in the Cell-DEVS simulation sub-model, the agent selects a cell randomly between the cells in the tie. This kind of randomization may affect the stability of the solution in the enemy section only, however it is not a serious problem as the enemy section is based on our assumptions.

4. CELL-DEVS AGENT SUB-MODEL

4.1. Real-time Cell-DEVS

The time advances in the DEVS and Cell-DEVS models based on the availability of the events. Thus, the simulation runs in virtual-time in which, after servicing every event, the simulation time advances to the next scheduled event time. To visualize the agent model, we need to run the agent model simulation in real-time, so that the events can be transferred to the visual engine, resulting in a real-life visualization of the battlefield. CD++ is designed and implemented based on the DEVS abstract simulation mechanism [21]. A Root Coordinator object acts as a coordinator with the top-coupled component in a CD++ model, which is responsible to advance the time to the next event time and also send and receive the I/O of the DEVS model.

We modified the Root Coordinator event scheduler function to work in real-time, in which the events are served at the time they are serviced and the time advances based on the wall clock time. We have added two new features to the CD++ simulator, which make CD++ capable of receiving DEVS inputs from the network and injecting them to the model, and at the same time sending outputs of the DEVS model to the network. A separate thread was added to the CD++ software structure to make it capable of listening to the network inputs without interrupting the main execution sequence. The input thread executes an added function of the Root Coordinator, which creates a network socket and listens to the network in a blocking mode. As soon as a network packet is received, the content of the packet is extracted and saved in the input bag of the Root Coordinator, which will service the input.

In order to send inputs to any specific atomic cell, we modified the CD++ MainSimulator post registration function, which is responsible to create the DEVS ports defined in the model file. In the modified version, the MainSimulator creates a default input port for each atomic cell (Figure 4). These ports are used later by the Root Coordinator to inject inputs

to the specific cell based on the coordinates indicated in the network message. Once an input is received, The Root Coordinator sends an input message to the input port of the Top coordinator, which is connected to the specific cell that is the destination of the message.

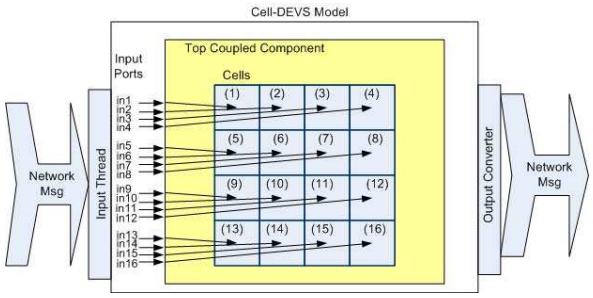

Figure 4. An example Cell-DEVS model structure and interfaces

To submit the changes of the cell value changes, we have added a function to the Root Coordinator, that extracts the outputs of atomic cells from the Y messages (the output carrying message defined in DEVS abstract simulation algorithm) and sends them to the network.

4.2. Global Message Structure

The collaboration of the sub-models 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 three types of messages:
   - The *dimension message* carries the size of the cell-space from the CellAgent sub-model to the 3D real-time visualization sub-model at the start of the execution.
   - The *cell-space update message* carries the cell value changes during the execution from the CellAgent sub-model to the 3D real-time visualization sub-model. It also carries the initial coordinates and personalities of the blue agent from the 3D real-time visualization sub-model to the CellAgent sub-model and the initial coordinates blue agent's flag from the 3D real-time visualization sub-model to the CellAgent sub-model.
   - The *visualization agent update message* carries the visualization changes during the execution from the 3D real-time visualization sub-model to the CellAgent sub-model. It also carries the initial coordinates of the red agent's flag from the CellAgent sub-model to the 3D real-time visualization sub-model and the initial coordinates blue agent's flag from the 3D real-time visualization sub-model to the CellAgent sub-model.
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. z: used to carry the layer axis value (the layer dimension or the layer coordinate).
5. v: used to carry the value of the cell(x,y,z).

These messages are embedded in a UDP packet, and transferred during the execution of the model through the network. The design of the system is such that the number of messages transferred through the network is a low as possible thus preventing delay in the message transfer.

4.3. Cell-DEVS Agent Definition Model

The basic element of our CellAgent sub-model is a VCELL Agent (VCELLA), which represents a primitive combat unit (tank, transport vehicle, etc.). The combat battlefield is represented in the CellAgent sub-model as a two-dimensional cell space as shown in Figure 5. Each cell in the space can be occupied by ***red*** agent of VCELLA. Each red agent can move to the next cell in the movement range or stay in the same cell. The sensor range is the area that is defined for each red agent to get the available number of friendly and enemy agents and their personalities values. The user defines the dimension of the combat battlefield and the initial state of red VCELLA agents at diagonally opposite corner of the red agents in the VisualAgent sub-model. Red flag is also positioned in the red VCELLA's corner. The goal for the red VCELLA is to reach the blue flag successfully. The Combat CellAgent sub-model is defined using the modified CD++ version descried in Subsection 4.1.

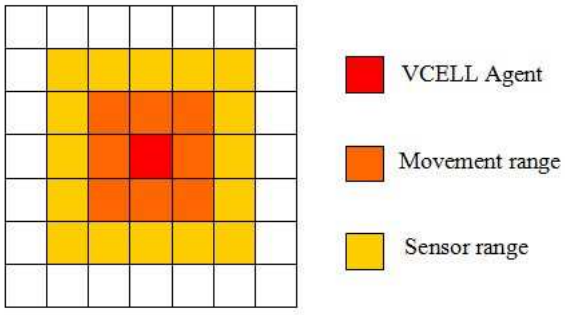

Figure 5. Movement and sensor range of VCELLA

Figure 6 illustrates the CellAgent sub-model in CD++ with the specifications and the network interface. The CellAgent sub-model is composed of *CD++ O/P Driver Component* and *CD++ Input thread*. The ***CD++ O/P Driver Component***

sends the battlefield dimensions to start up the VisualAgent sub-model. Then it sends the initial and updated values of red agents, their weights, and the red flag position in real-time to the VisualAgent sub-model. The ***CD++ Input thread*** receives the initial and updated values of blue agents, their weights, and the blue flag position in real-time from the VisualAgent sub-model. Then it invoked these values in the cell space of the model.

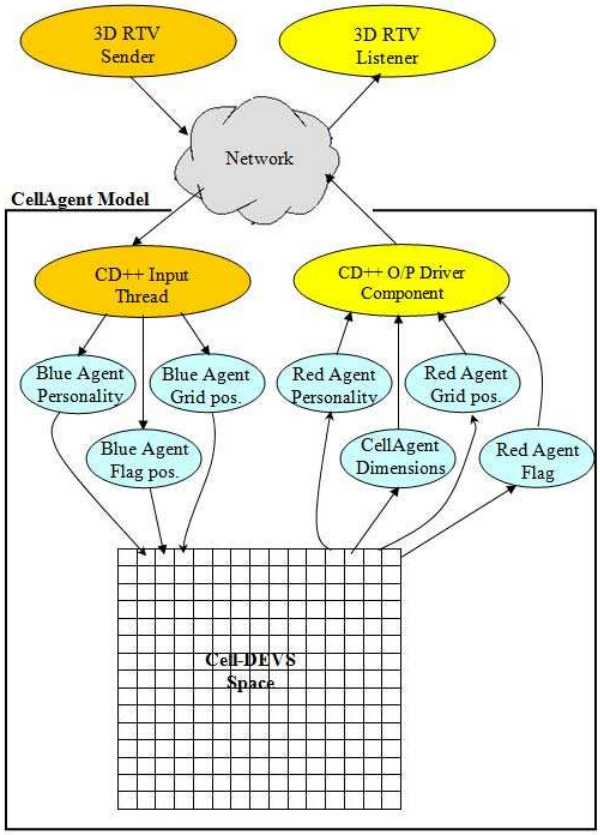

Figure 6. CellAgent model implementation in CD++

The model file of CellAgent sub-model reads the initialization data from a CD++ associated value file. The value file has the initial values of the red agents, their weights, and the red flag position. The dimensions of the battlefield is defined in the model file. The model file is composed of six layers. The first layer contains the red agents, the second layer has their weights, and the third one contains the red flag position, which gets its initial values from the value file. The other three layers are allocated to the blue agents, their weights, and the blue flag position, which get its values from the 3D real-time simulation sub-model.

The personality of VCELLA can be defined by the weight towards enemy agents and the weight towards the enemy flag which specify how VCELLA interacts with information within its sensor range. Each VCELLA has one of three states: alive, injured, or killed. The health state, $0 \leq H \leq 1$, is the measure of an agent's health. The agent's health can be defined as shown in equation (3):

$$H = H * \sigma\left(\frac{1+F-E}{F}\right) \tag{3}$$

where

F Number of friendly entities within sensor range;
E Number of enemy entities within sensor range;
σ function of x as defined in (4);

$$\sigma(x) = \begin{cases} 1, & 1 < x \\ x, & 0 \leq x \leq 1 \\ 0, & \text{otherwise} \end{cases}, \tag{4}$$

As discussed earlier, the CellAgent sub-model can interact with the 3D visualization in real-time. We have also added a generic interface to the simulation engine enabling it to interact with the external environment (e.g. 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 discussed in Subsection 4.2.

5. 3D REAL-TIME VISUALIZATION (RTV)

3D Visualization of combat can provide a number of benefits. First, it provides decision makers with an interactive environment to verify the accuracy of these models by comparing the results of an actual combat with the output of a simulated version. Once the model is validated, it can then be used to predict the behavior of an existing combat. Displaying these predictions in a visually informative manner allows the decision makers to understand a view of the situations and their soldiers and warfare in order to make decisions that are more effective. Furthermore, interactive simulation along with the 3D visualization allows trainers to apply different combat tactics and enemy behaviors. While real training would be risky and costly to perform, these risks can be minimized by simulating untested approaches first. 3D visual interfaces provide a more understanding of interaction. Additionally, high-fidelity graphics enables an observer in better comparing a simulated combat to a traditional 2D visualization.

5.1. Visualization sub-model Description

The 3D real-time visualization is used to visualize the simulation output results of the CellAgent sub-model and also to implement a collaborative model which shares its components on two different simulation engines. The visualization renders the red agents, and the creation of the blue

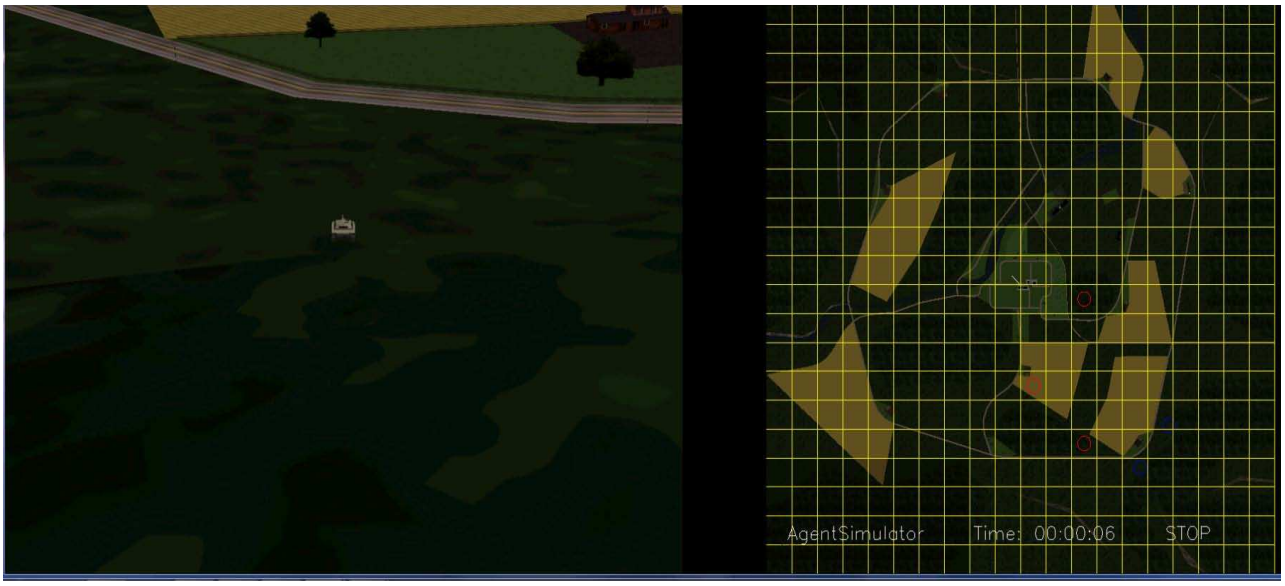

Figure 7. 3D Real-time Visualization view

agents and their characteristics based on their movement algorithm and their personalities. The 3D real-time visualization model is implemented using Vega Prime and OpenGL [14]. Vega Prime is a high-performance software environment and toolkit for real-time 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 C++ callable functions.

In the 3D real-time visualization sub-model, the combat can be seen in a 3D view for both red and blue agents. The blue agent personality calculations and position updates are done in the RTV sub-model based on the received data of the red agents. The red agent's personalities and positions are received from the CellAgent sub-model in real-time. The CellAgent sub-model receives the blue agent's personalities and positions from the RTV in real-time.

The 3D scenes are rendered using 3D Openflight models. The terrain model consists of trees, buildings, roads, etc. The agents are represented by a 3D Tank model. We can control the environment effects and the time of the day in the 3D scene visualization. A 3D scene is shown in a window that is divided into two channels; one for perspective view of 3D scene (on the left), and the other for the orthographical view of the 3D scene which acts as 2D Map of the area (on the right) as shown in Figure 7.

On the perspective view of the first channel, a 3D model for each agent (3d tank object) is displayed in the 3D scene. The 3D scene 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 object.

On the orthographical view of the second channel, a yellow grid is created representing the cellular grid of the combat simulated area. The red agent's positions received from the CellAgent sub-model are rendered by red circles and the blue agent's positions represented by a blue circle (see Figure 7). The orthographical view is capable of zooming in and out and the cellular grid can be removed for a better view.

5.2. RTV Sub-model Implementation

Figure 8 illustrate the hierarchy of the 3D real-time visualization sub-model, which was implemented in Visual C++, and it consists of three main components:

1. The *RTV Listener*, which receives the data of the red agents from the CellAgent sub-model.
2. The *RTV Visualization*, which is responsible of creating of the blue agents and the display of the 3D visualization scene.
3. The *RTV Sender*, which sends the data of the blue agents to the CellAgent sub-model.

The ***RTV Listener*** is a separate thread, which is spawned for receiving the red agent's data. First, the *RTV Listener* is responsible for receiving the dimensions of the cell-space from the CellAgent sub-model in order to start the *RTV Visualization* to render the 3D scene. Then, the *RTV Listener* receives the red agent's flag position. Finally, the *RTV Listener* receives the red agents grid positions updates and their personalities in real-time from the CellAgent sub-model.

The ***RTV Visualization*** is the main part of the 3D real-time visualization sub-model. The *RTV Visualization* is responsible of the setting up the 3d visualization of 3D real-time visualization sub-model.

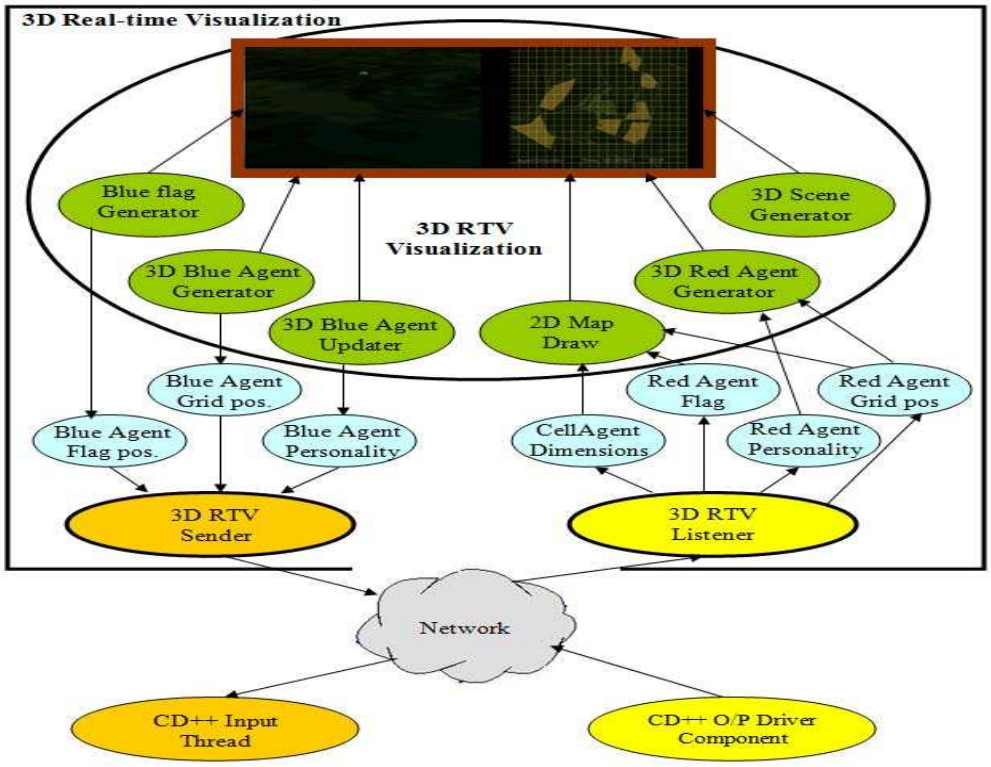

Figure 8. 3D Real-time Visualization Hierarchy

The *RTV Visualization* is composed of six main modules:

1. ***3D Scene Generator***, which is responsible for the set up and synchronization of the 3D scene, drawing of different 3D objects (terrain, tanks, buildings, etc), defining and controlling different environments effects (daytime, clouds, sun, etc), and drawing the other modules. The 3D Scene Generator also removes the dead agents from the 3D scene.
2. ***3D Red Agent Generator***, which creates a 3D object for the red agents. It positions the red agents based on the received coordinates from the *RTV Listener*. The 3D Red Agent Generator sets and updates the red agents with their personalities which are received from the *RTV Listener*.
3. ***2D Map Draw***, which consists of:
   - ***DrawGrid***, which receives the cellular space dimensions from the *RTV Listener* and draws it on the 2D Map channel.
   - ***DrawCircle***, which gets the agent's positions and draws the red and blue circles according to the agent's type at the corresponding coordinates.
   - ***DrawFlag***, which draws a box for each flag according to its color at the corresponding coordinates.
4. ***3D Blue Agent Generator***, which creates the blue agents (which include the 3D object model, personalities, and position). The 3D Blue Agent Generator sends the position of the blue agents to the *RTV Sender*.
5. ***3D Blue Agent Updater***, which calculates different personalities for the blue agents according to the received data for the red agents. Then it sends the personalities of the blue agents to the *RTV Sender*.
6. ***Blue Flag Generator***, which creates a 3D object for the blue agent's flag at the user defined position, then it sends the position to the *RTV Sender*.

The ***RTV Sender*** is a separate thread, which is spawned for sending the blue agents data. First, The *RTV Sender* sends the grid position of the blue agent's flag to the CellAgent sub-model running on the CD++ workstation. After that, it sends the blue agents grid positions and their personality updates to the CellAgent sub-model while the model develops.

Finally, the 3D real-time visualization sub-model is capable of deploying different Openflight 3D terrain models and different cellular areas (dimensions and initial values) without changing the code of the visualization. Some videos are generated using the proposed model that can be found in [29, 30]

6. CONCLUSIONS

We present a 3D real-time visual Cellular Agent model (VCELL) for collaborative of Cell Agent simulation with 3D visualization in real-time of different battlefield combat scenarios. This work is done using a 3D real-time visual engine and the CD++ simulator running Cell-DEVS models. The VCELL model is not only used for prediction, but also to improve the understanding and learning process in a land combat. VCELL is designed to help analysts and trainers to get maximum gain by interactively simulating the scenarios, validating the results and training the soldiers in an effective environment. This work incorporates two sub-models: a Cell-DEVS agent simulation and a 3D visualization agent. The two sub-models provide model reusability and interoperability allowing for integration or replacement of any of them. The Visual Cell-DEVS Agent of a land combat is a new way of enhancing the randomization movement problem of movement algorithms in agent-based simulation by using the 3D visualization agent which gets the real position of agents in the combat. The Cell-DEVS agent simulation reduces the time development taken to create or modify tactical scenario in the 3D visualization simulation by using Cell-DEVS formalism which has an interpreter to write simple rules. These rules are transformed to be visualized in the 3D visualization by using 3D scenario generation. To implement the Cell-DEVS communication interface, we modified the CD++ simulator core engine. Our model is implemented using robust software tools to make the real-time visual Cell-DEVS agent model more flexible and more scalable.

REFERENCES

[1] A. Ilachinski. Irreducible semi-autonomous adaptive combat (ISAAC): An artificial life approach to land combat. Technical Report CRM 97-61, Center for Naval Analyses, Alexandria, VA, Aug. 1997.

[2] A. Ilachinski. Irreducible semi-autonomous adaptive combat (ISAAC): An artificial life approach to land combat. *Military Operations Research*, 5:29–46, 2000.

[3] M. K. Lauren. Modeling combat using fractals and the satistics of scaling ssystems. *Military Operations Research*, 5(3):47–58, 2000.

[4] G. A. Cowan, D. Pines, and D. Meltzer. *Complexity : Metaphors, Models, and Reality*. Addison-Wesley, 1994.

[5] S. Kauffman. *At Home in the Universe: The Search for Laws of Self-Organization and Complexity*. Oxford University Press, 1995.

[6] C. G. Langton. *Artificial Life: An Overview*. MIT Press, 1995.

[7] K. Mainzer. *Thinking in Complexity: the computational dynamics of matter, mind, and mankind*. Springer, 2004.

[8] A. Ilachinski. *Enhanced Isaac Neural Simulation Toolkit (EINSTein), an Artificial-Life Laboratory for Exploring Self-Organized Emergence in Land Combat*. Center for Naval Analyses, Beta-Test Usersí Guide CIM 610.10., 1999.

[9] G. White. The mathematical agent a complex adaptive system representation in bactowars. In *First workshop on complex adaptive systems for defence*, 2004.

[10] M. K. Lauren and R. T. Stephen. Map-aware non-uniform automata - a new zealand approach to scenario modelling. *Battlefield Technology*, 5(1):27–31, 2002.

[11] M. Barlow and A. Easton. Crocadile: An open, extensible agent-based distillation engine. *Information & Security*, 8(1):17–51, 2002.

[12] A. W. Gill. Improvement to the movement algorithm in the mana agent-based distillation. *Battlefield Technology*, 7(2):19–22, 2004.

[13] Ang Yang, Hussein A. Abbass, and Ruhul Sarker. Wisdom-ii: A network centric model for warfare. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 173–173. Springer Berlin / Heidelberg, 2005.

[14] A. W. Gill and D. Grieger. Validation of agent based distillation movement algorithms. Technical report, Defence Science and Technology Organization, Australia, 2003. DSTO-TN-0476.

[15] Gabriel A. Wainer and Norbert Giambiasi. *Timed Cell-DEVS: modelling and simulation of cell spaces*. Springer-Verlag, 2001.

[16] Dion Grieger. Comparison of two alternative movement algorithms for agent based distillations. Technical Note DSTO-TN-0777, Defence Science and Technology Organisation Edinburgh (Australia) Land Operations Div, Aug. 2007.

[17] Ang Yang, H.A. Abbass, and R. Sarker. Characterizing warfare in red teaming. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(2):268 –285, 2006.

[18] F. W Lanchester. *Aircraft in Warfare: The Dawn of the Fourth arm*. Constable and Company Limited, London, 1916.

[19] D. S. Alberts and T. J. Czerwinski. *Complexity, Global Politics, and National Security*, chapter 9, pages 99–111. National Defense University, Washington, D.C., 1997.

[20] R. A. Brooks and L. Steels. *The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents*, chapter 2, pages 25–81. Lawrence Erlbaum Associates, Hillsdale, NJ, 1995.

[21] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000.

[22] S. Wolfram. *Theory and applications of cellular automata(Advances Series on Complex Systems), Volume 1*. World Scientific, 1986.

[23] Gabriel A. Wainer. CD++: a toolkit to define discrete-event models. *Software, Practice and Experience*, 32(3):1261–1306, November 2002.

[24] Mark Griffiths. The educational benefits of videogames. *Education and Health*, 20(3):47 – 51, 2002.

[25] Training Circular 25-20. *A Leader's Guide To After-Action Reviews*. Department of the Army, Washington, DC., September 1993.

[26] Committee on Modeling, Simulation, and Games. *The Rise of Games and High Performance Computing for Modeling and Simulation*. The National Academies Press, 2010.

[27] Marc Prensky. *Digital Game-Based Learning*. Paragon House, 2007.

[28] PRESAGIS. Vega prime, March 2011. `http://www.presagis.com/products_services/products/ms/visualization/vega_prime/`.

[29] `http://www.youtube.com/watch?v=4qUBRrFIYKA`.

[30] `http://www.youtube.com/watch?v=hgN4iXkkXfg`.