SERIOUS GAME DEVELOPMENT METHODOLOGY VIA INTEROPERATION BETWEEN A CONSTRUCTIVE SIMULATOR AND A GAME APPLICATION USING HLA/RTI

Changbeom Choi,[a] Moon-Gi Seok,[b] Seon Han Choi,[c] Tag Gon Kim,[d] and Soohan Kim[e]

[a] [b] [c] [d] Dept. of Electrical Engineering, Korea Advanced Institute of Science and Technology
[b] Video Display Biz. Technical Planning Group, Samsung Electronics HQ

[a]cbchoi@smslab.kaist.ac.kr, [b]mgseok@smslab.kaist.ac.kr, [c]shchoi@smslab.kaist.ac.kr, [d]tkim@ee.kaist.ac.kr, [e] ksoohan@samsung.com

ABSTRACT

This paper proposes a serious game development methodology that utilizes interoperation between an existing virtual world application and constructive simulators. For time synchronization and data conversion between them, the proposed methodology is comprised of three specified processes: game loop analysis, game agent design and development, and parameter tuning. We use a High-Level Architecture (HLA) to ensure interoperation. By interoperating a constructive simulator with an existing virtual world application, a serious game developer can save effort by extending a serious game application, rather than building a serious game from scratch. In addition, trainees can obtain more realistic experiences.

Keywords: Interoperation, System of Systems, Constructive Simulator, Serious Game, Virtual Military Training

1. INTRODUCTION

With the impressive growth of the game industry over the last several decades, serious games have emerged to educate and train learners, rather than provide entertainment. Serious games allow learners to experience situations that are impossible in the real world due to safety, cost, and/or time. For this reason, the game industry has developed various types of serious games, including games for military, manufacturing, and medical purposes.

In the military field, several commercial serious games, such as Virtual Battle Space 2 (VBS2), Military Open Simulator Enterprise Strategy (MOSES), and Delta3D, are already available in the market. The main purpose of these games is to simulate military situations, and they fundamentally allow users to edit terrain and scenarios to create specific war environments. Nevertheless, a problem with these serious games is the limited accessibility granted to their operating systems; thus, these games only enable war scenarios and terrain within a limited scope (Gwenda, 2004). In addition, these games are based on game engines that lack modifiable script languages. Therefore, the creation of new war scenarios, modeling of combat entities, and reuse of such entities are greatly hindered, which results in a failure to adapt expandable war scenarios (Part et al., 2010). Such limitations restrict more detailed and expandable representations of military simulation development.

Our approach, therefore, overcomes the precedent limitations of utilizing existing military serious games for expandable war scenarios. To this end, we have paid attention to separating game applications from scenario interpreters. Game applications are existing military serious games, such as VBS2, MOSES, and Delta3D, whereas scenario interpreters are constructive simulators that generate dynamic situations based on the users' requests. Specifically, users generate war scenarios through a constructive simulator, and the simulator sends the scenario to the game application for battlefield visualization. While users conduct training through the scenario within the game application, mutual interactions frequently occur between the game application and the constructive simulator. Accordingly, the key issue for this approach is the method of interaction between these two separated parts.

The Federation Development and Execution Process (FEDEP) is a standardized process for developing interoperable systems. Because FEDEP is a general-purpose process that needs to describe two specific kinds of systems (the game application and the constructive simulator) and represent the characteristics of their interactions, we have customized the existing FEDEP and propose the new Military Serious Game Development and Execution Process (MSGDEP). The primary purpose of the MSGDEP is to provide not only a process, but also facilities that assist the existing game application in utilizing expandable war scenarios.

Thus, in this paper, we propose the MSGDEP for interoperation between game applications and constructive simulators. Specifically, the proposed methodology centers on two ideas: 1) time synchronization and 2) data conversion. To satisfy both ideas efficiently, the MSGDEP is comprised of three specified processes: game loop analysis, game agent design and development, and parameter tuning. To

interoperate between the game application and the constructive simulations, we use a High-Level Architecture (HLA), which is used for distributed computer simulation systems. In our empirical study, we achieved time synchronization and data conversion based on the HLA (IEEE 1516-2010). By interoperating the constructive simulator with the existing virtual world application, serious game developers can save effort by extending a serious game application, rather than building a new serious game from scratch; in addition, trainees can acquire more realistic experience.

As a case study, we built and developed a military training scenario for a Nuclear/Bio-Chemical (NBC) situation. The outcomes of the case study will show the usefulness of the proposed work, such as how the flexibility and reconfiguration of the war game scenario improve, as well as how effectively the users can train within the scenario. The successful execution of this study can offer an immediate application for military training, and is particularly suited to war scenarios based in the Korean Peninsula.

The rest of the paper is organized as follows: Section 2 introduces existing military game applications and FEDEP. In Section 3, we explain the proposed SGDEP via interoperation between a game application and constructive simulators. Section 4 illustrates a case study that incorporates the proposed methodology, and finally, Section 5 concludes this study and proposes future extensions for a more complete solution.

2. RELATED WORKS

In this chapter, we will first introduce the existing serious games and describe FEDEP, which is a standardized and recommended process.

2.1. Virtual Battlespace 2 (VBS2)

The VBS2 is a comprehensive, open platform that uses gaming technology to provide tactical training experience and mission rehearsals (Virtual Battlespace 2, 2013). Several case studies have shown that VBS2 provided an immersive experience to a trainee through lifelike virtual environments. VBS2 provides two methods to extend its platform: integration and interoperation. First, VBS2 provides a plug-in interface for developers, so that other simulation systems can be coupled tightly with VBS2. Second, VBS2 allows for interoperation between various simulation systems via the DIS protocol or HLA. Therefore, in order to extend VBS2, the developer may choose between integration and interoperation. When using the integration method, the developer should understand the game loop of VBS2, so that the simulation system can be tightly integrated into VBS2. On the other hand, in order to extend VBS2 via interoperation, VBS2 participates in the federation and interoperates with other simulation systems. However, to the authors' knowledge, no methodology has been proposed to support interoperation between VBS2 and an existing constructive simulator. For our research, we modified the existing federation development methodology. By clarifying the requirements for each development phase, a developer can define the shared information between a serious game and a constructive simulator, and implement them easily.

2.2. Military Open Simulator Enterprise Strategy (MOSES)

The US Army Research Laboratory Simulation and Training Technology Center (ARL-STTC) developed a virtual world application called the Military Open Simulator Enterprise Strategy (MOSES) for military training needs (Maxwell et al., 2012). In order to develop a flexible virtual training framework, the ARL-STTC conducted research that utilized gaming and virtual world technology. To develop a flexible virtual training framework for trainers and trainees, the framework needed to allow for variable fidelity, based on the training objectives. MOSES is based on the Open Simulator, which is an open-source project to provide a virtual world server that can be accessed via the same viewer as SecondLife (OpenSimulator, 2013)

Similar to SecondLife, users of MOSES can upload and present content, such as buildings, objects, or training content, into the virtual world. Moreover, every object in the virtual world is interoperable and may have various scripted, interactive behaviors. In other words, the virtual world server has a script engine that allows the user to upload a script, which contains the behavior of an object, to the server. Therefore, when a user interacts with an object in the virtual world, the script engine interprets its script, executes actions, and represents them to the users via a virtual world viewer. Such functionality enables trainers to develop flexible training content. Trainers can arrange the positions of buildings or place an object in the training field. Afterwards, trainers can build scripts for each object to determine its behavior when a trainee interacts with it during the training course.

However, MOSES, as well as other virtual world applications, has limitations on extending training content. In particular, the script engine does not support the creation of training courses that are based on accurate simulation results. Yet, if a trainer wants to build a realistic training course for an evacuation process, the script engine should support the realistic simulation of the target environment or the systems, such as the propagation of a chemical cloud after a bomb detonates or the propagation of a chemical cloud based on geographical features and environmental factors. Moreover, even if the script engine can support a realistic simulation, the computation of such a realistic simulation can burden the application servers, so that the servers cannot service the trainees.

2.3. Federation Development and Execution Process

In the modeling and simulation fields, HLA has been approved as an IEEE standard to specify interoperating, heterogeneous simulations within

distributed environments. In this standard, a standalone simulator is called a federate, and the set of federates that comprise a larger system to achieve the same purpose is called a federation (IEEE 1516-2010). If a simulator is compliant with HLA protocols, we call it an HLA-compliant simulator. The Federation Development and Execution Process (FEDEP) is a recommended development process used to develop HLA-compliant simulators and federations (IEEE 1516.3-2003). FEDEP is a standardized and recommended process for developing interoperable, HLA-based federations. Figure 1 shows the phases of FEDEP.

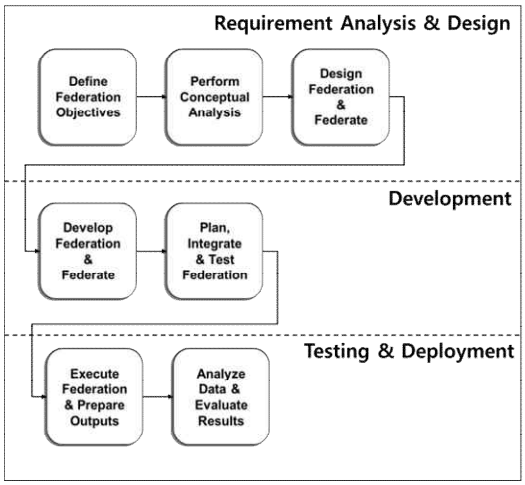

Figure 1: Phases of Federation Development and Execution Process

As shown in Figure 1, a developer should first define the objectives and requirements of the federation and confine the scope of the federation's development to the identified requirements. When the objectives of a federation are fixed, a developer should perform a conceptual analysis of the target system. Then, the developer will design and implement the federation and each federate. In this phase, the developer should identify the input/output data of each federate, in order to create the Simulation Object Model (SOM). The SOM contains the types of data that the simulator will exchange during the simulation. The Federation Object Model (FOM) is the set of SOMs that constitutes the federation data. After identifying the dataset, the developer has three options: utilizing an existing federate, develop a federate from scratch, or modify a legacy simulator into an HLA-compliant simulator. After the federates are implemented, the developer may integrate them into a federation and test it. After integration and testing, a user executes the federation and analyzes the data. Finally, the federates are revised based on the analyzed results.

However, the FEDEP is insufficient for developing a federation among the serious game and the constructive simulations for several reasons. First, a serious game has usually been implemented already; thus, it is almost impossible to modify the game application to support HLA protocols. Second, the time units of the serious game and the constructive simulator may be different; thus, the developer must tune the time resolution between them. For example, the default time unit of a constructive simulator may be hours, but the default time for a serious game may be milliseconds.[1] Therefore, time synchronization between a serious game and constructive simulators is different from interoperation between simulators. Third, standard distance values that differ between the serious game and the constructive simulator should be calibrated. For example, the standard distance value of a constructive simulator can be in kilometers, and the space of the training ground can be 100 m² or more. However, such a training ground will hinder the training experience in a serious game. Usually, the designer of a virtual training ground wants to maximize training; therefore, training grounds are usually relatively small and bounded. Therefore, the developer should consider and regulate the values between the constructive simulator and the serious game.

In the next section, we will propose a methodology for interoperation between a serious game and a constructive simulator that takes the aforementioned problems into consideration.

3. PROPOSED SERIOUS GAME DEVELOPMENT METHODOLOGY

Before moving to the central part of the MSGDEP, we must identify the components of the SGMT and their roles. The proposed SGMT consists of an existing serious game that provides virtual battlefield situations for training and several constructive simulators to describe the situations in detail. Let us suppose that trainees exercise MOUT (Military Operations in Urban Terrain) using the proposed SGMT. In this case, the existing serious game provides battlefield situations, such as the number of soldiers and the constructions that are involved, while the constructive simulators compute numerical calculations, such as atmospheric diffusion and damage assessment. During a simulation, the calculations of the constructive simulators are reflected in the serious game. Consequently, the separation between the existing serious game and constructive simulators enables to reuse individual components, and trainees can experience expandable battlefield situations easily by communicating with various constructive simulators in the existing serious game. From the viewpoint of system engineering, the SGMT is considered to be a system of systems (SoS). Therefore, developing a federation that consists of a serious game and a constructive simulator and building a system of systems are alike.

In our previous research, we proposed a System of System Entity Structure (SoSES) and Federate Base (FB) framework to manage federates and synthesize the federation (Kim et al., 2013). When a developer wants

[1] The time unit of a constructive simulator is logical time; thus, the designer of the simulator can decide the unit time of the simulator.

to build a federation, the SoSES/FB framework supports the developer in synthesizing the federation, based on its objectives. The SoSES denotes the structure of the federation and helps the developer to choose which federate will join the federation. After the user selects a federate, the framework automatically bring federates from the FB and synthesizes a federation from them. In other words, the SoSES is a blueprint of a federation, and the FB is a repository for federates.

However, SoSES/FB is not suitable for developing or extending a serious game via interoperation. Unlike typical federation development, SGMT development should consider the user's behavior and the time synchronization between a game and its simulators. Generally, when the designers of a SGMT build a virtual training field, they arrange the virtual objects to maximize the training experience. As a result, the size of a virtual training field is relatively small, and the placement of the virtual objects leads the user to acquire virtual training experience. On the other hand, the objective of a constructive simulator is to acquire reliable simulation results from the simulation models. Therefore, the developer should narrow the gap between the serious game and the constructive simulator, in order to build and extend the SGMT via interoperation.

Figures 2 and 3 show our proposed development methodology. First, the developer should consider the objective of the federation and perform conceptual analysis. During the conceptual analysis, the developer must consider which serious game application and constructive simulators should form a federation. In this phase, the developer decides to develop a game agent or federate from the beginning or utilize existing federates from the FB. Figure 2 shows the former process, and Figure 3 shows the latter process. The differences between FEDEP and the MSGDEP can be characterized by the federation synthesis, game agent development, and parameter tuning phases. In the following section, we will explain each phase in detail.

3.1. Federation Development Process for SGMT

As shown in Figure 2, the proposed development methodology extends the FEDEP. The differences between the FEDEP and the federation development process for SGMT are in the game loop analysis, game agent design, game agent development, and parameter tuning phases.

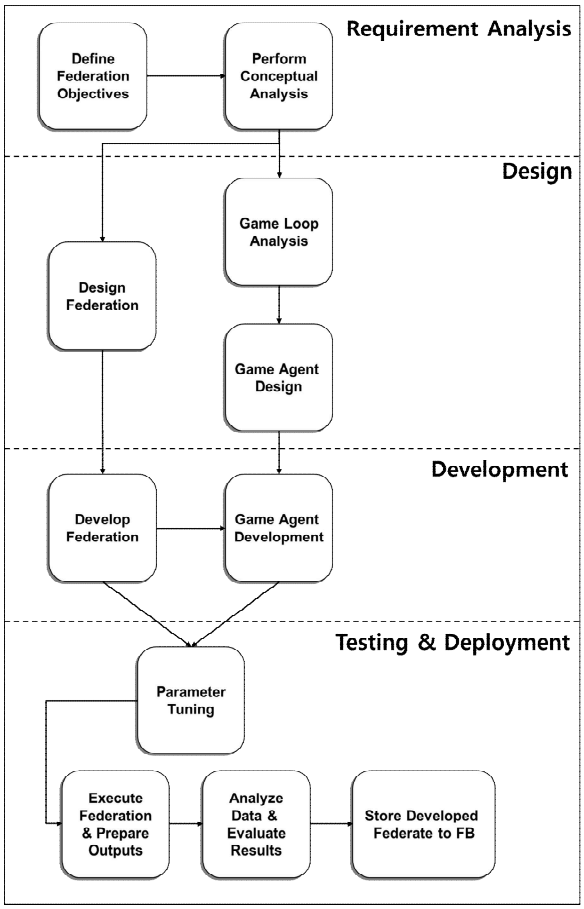

Figure 2: Federation Development for Interoperation Between Serious Game and Constructive Simulator

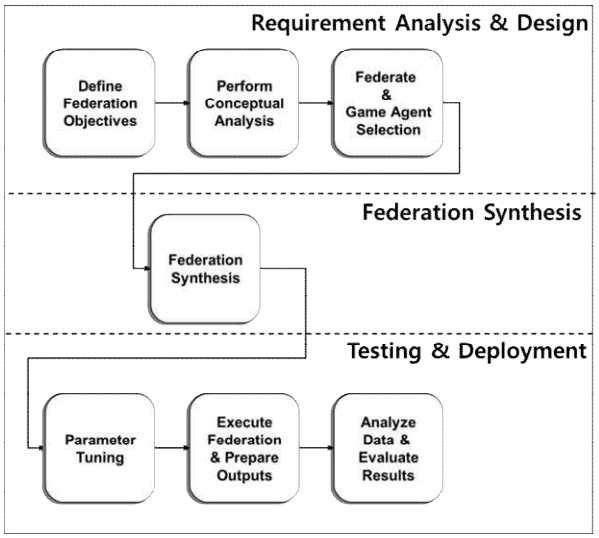

Figure 3: Federation Synthesis for SGMT Development Using HLA/RTI

3.1.1. Game Loop Analysis

In order to interoperate a constructive simulator and a given game application, the developer should identify the necessary information for the constructive simulator and game application. For example, the constructive simulator should know the position of the

user participating the virtual training, and the game application should know the states of the users, which are determined by the constructive simulators. In order to acquire such information, the developer should understand the *game loop* of the game application. As described by Valente et al. (2005), input data acquisition, data processing, and rendering occurs simultaneously while the game is running; in order to handle the process, the game loop is made up of the *read player input*, *update*, and *render* stages.

Therefore, in order to interoperate a serious game application and a constructive simulator, the simulation results from the constructive simulator should be reflected before the render stage. In order to reflect the simulation results before the render stage, the developer has two options: modify the server structure of the serious game or modify the client program of the serious game. For example, the former option may involve inserting additional game logic into the update stage, while the latter option may reflect the simulation results during the read player input stage. Between these two options, the former option may be more suitable for implementing interoperation features into the serious game; however, the latter option may be more suitable for cases in which the server and the client of a serious game have already been developed.

In this study, we assume that the client and the server of the serious game have already been developed. To tackle this problem, we built a special client for a serious game application called the Serious Game Agent (SGA), so that the client subrogates the constructive simulator to reflect the simulation results to the serious game. Therefore, in the game loop analysis phase, the developer should understand the protocol between the server and client of the serious game.

3.1.2. SGA Design/Development

When the analysis of the game logic of a serious game application is finished, the developer should design and develop the SGA. As mentioned earlier, the SGA is a gateway for the game to interchange information between constructive simulators and a game. The objectives of the SGA are to manage the mapping between the information from the serious game and the information from the constructive simulator, and transfer the information to the other side as quickly as possible. Therefore, the developers of the SGA may focus on how information is managed between the game and simulators, rather than rendering the objects in the serious game. Figure 4 shows the architecture of the SGA. The HLA/RTI controller of the SGA handles the communication between the HLA/RTI and the SGA. In particular, the HLA/RTI controller controls the invocation of HLA services and handles the HLA service callbacks. Correspondingly, the service protocol between the server and the client of the serious game is implemented in the serious game connector. Then, the SGA transfers information from the constructive simulator to the serious game, based on the information mapping tables, and vice versa. Finally, when the development of the game agent is finished, the developed game agent is stored and federates to the FB.

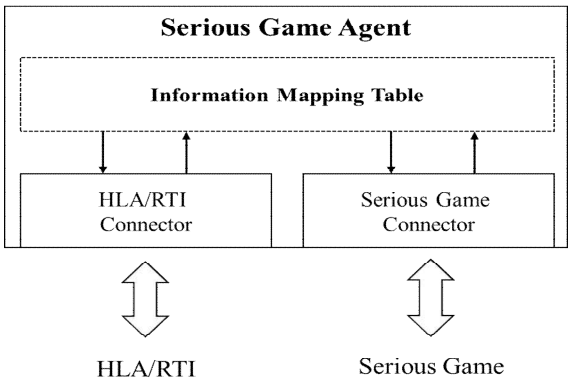

Figure 4: Architecture of a Serious Game Agent

3.1.3. Parameter Tuning

Since the game application and the constructive simulators are different, the developer should tune the parameters. Before we discuss this phase, we need to analyze the characteristics of the serious game and the constructive simulators. The objectives of a constructive simulator are to measure and analyze the performance index of a simulation model. A developer designs and implements the constructive simulator to obtain reliable simulation results. Therefore, the simulation time and simulation space must reflect the real world.

However, the scales of time and the space are relative to the users. For example, the speed of a vehicle in the simulator may be denoted as km/h, which is important because the data affects the simulation results. On the contrary, the trainers of a serious game will not consider the exact speed of a vehicle; they may regard the relative speed as more important. Moreover, the distances between objects may differ. If the simulator uses a different distance scale in the serious game, the simulator may generate unintended simulation results. In contrast, if the serious game utilizes the distance scale of the simulator, the trainee may become bored, because implementing a training field with real scales will generate an enormous virtual training field. As a result, the developer should consider the scales of time and space and tune the parameters iteratively, until the requirements and implementation of the federation are met.

3.2. Federation Synthesis Process for SGMT

As shown in Figure 3, the differences between the federation development process and the federation synthesis process for SGMT are in the *federate and game agent selection* and *federation selection* phases. The management structure of the federation and synthesis algorithm was detailed by Kim et al. (2013). After the selection and synthesis phases are finished, the developer should tune the parameters.

4. CASE STUDY: NUCLEAR/BIO-CHEMICAL EVACUATION TRAINING SIMULATOR

This chapter will detail our empirical research. In order to generate dynamic situations during serious gaming, we utilized the virtual world application In-World Editor as a serious game and the chemical diffusion simulator as a constructive simulator. First, we will introduce the serious game application and the constructive simulator. Then, we will share our experience about interoperating both of them. Finally, we will share what we learned during our empirical research.

4.1. In-World Editor

In-World Editor is a virtual world application based on the Unity 3D Engine and Photon server application (Unity 3D, 2013; Photon Network Engine, 2013). In order to provide a sense of reality within a well-built virtual training environment, the user can rearrange the objects during gameplay. Moreover, the application supports scripts, which allow objects in the virtual world to interact with the users. In addition, it supports interactions between multiple users. Each user shares a virtual training environment and trains with other users through each client. They can allocate virtual objects to the field and arrange the positions of objects that other users have allocated. Figure 5 shows a screen capture of the In-World application.

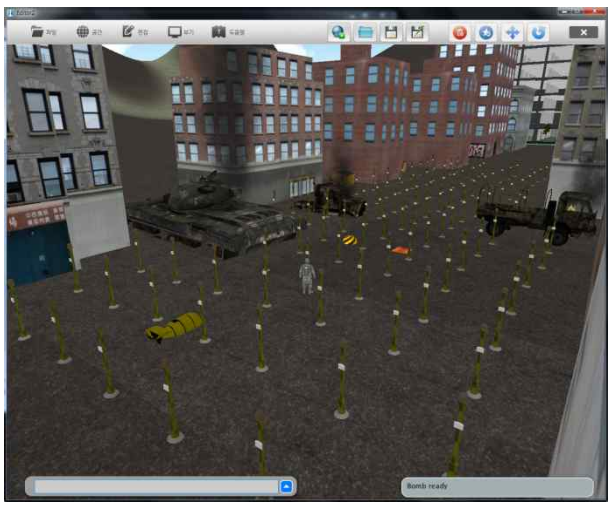

Figure 5: In-World Editor

To provide an immersive experience for users, this application provides some functionality to build virtual training environments. First, the user can allocate and remove various objects freely, such as buildings, cars, trees, sensors, bombs, and so on. Figure 6 shows the object allocation in a virtual training environment. The server of In-World Editor manages the assets, and the client shows them when the user of In-World Editor wants to allocate virtual objects to the virtual world.

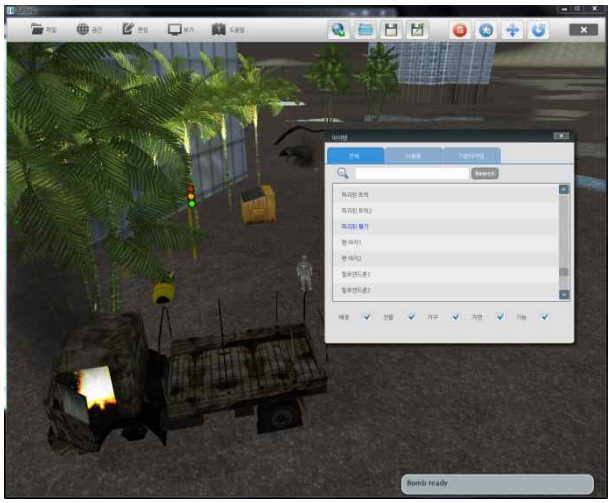

Figure 6: Object Allocation

In addition, users can interact with the allocated virtual objects. For example, users cannot go through obstacles that have been allocated onto a road. Therefore, the trainer can lead the trainee to the training content. Moreover, In-World Editor supports allowing the trainer to plant a bomb, and the trainer can detonate the bomb at any time. Using these objects, a trainer can create a well-built virtual training environment. Additionally, In-World Editor supports administrative functionality. The trainer can use script commands to control the virtual environment.

Figure 7: Interact with Object (detonate bomb)

In-World Editor provides interaction between users and the virtual world through 3D graphics; however, it cannot provide realistic simulation results to trainees. For example, if the trainer wants to build a training scenario in which the trainee must handle an evacuation due to chemical warfare, the developer should modify or insert the game logic for chemical warfare. In order to extend functionality without changing any of the game logic for In-World Editor, we utilize the chemical diffusion simulator by interoperating them.

4.2. Chemical Diffusion Simulator

The chemical diffusion simulator is a constructive simulator that calculates the distribution of the chemical compounds over the various geographical features. To obtain a realistic distribution of the compounds that considers the effects of solid walls and wind, the constructive simulator utilizes a numerical Computational Fluid Dynamics (CFD) model (Blazek & Jiri, 2001). Utilizing CFD models, the user of a constructive simulator can analyze various distributions of chemical compounds after a chemical detonation. Figure 7 shows a screen capture of the constructive simulator calculating the distribution of the chemical compounds.

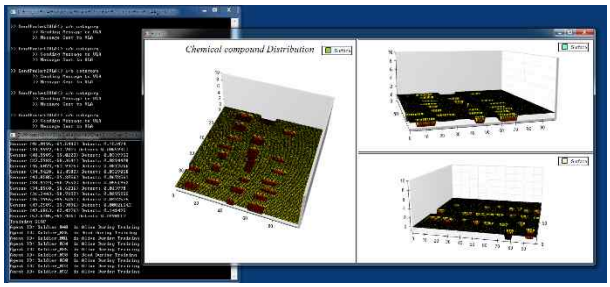

Figure 8: Chemical Compound Simulator

The CFD model discretizes the virtual space of the game into grids and solid boundaries and then computes the states of the grids iteratively based on the governing equations, boundary conditions, and states of the neighboring grids, as the simulation time advances. In the chemical diffusion simulator, the CFD model uses Roe approximate Riemman solvers to update the states of the grids, such as their density, velocity, and energy, based on the Euler equation for the governing equation, as well as solid walls and characteristic boundary conditions for the boundary condition, as seen in Figure 9 (Roe, 1981). When a chemical bomb explodes in the constructive simulator, the state of the grid where the exploded bomb is located changes, and the density of the chemical compound increases. From that point, the updated states influence the states of all neighboring grids during iterative computing of the CFD models. The distribution of chemical compounds is calculated based on the chemical compound's density and pressure, and may vary according to the bomb type and the environment.

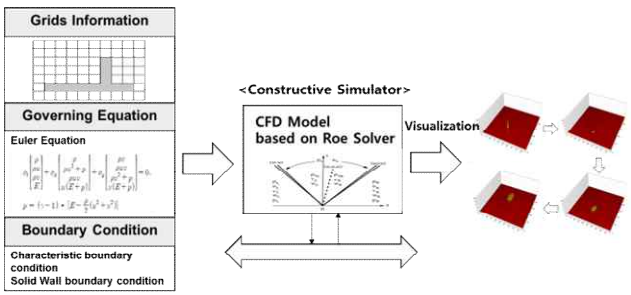

Figure 9: Boundary Condition for the Simulator

4.3. Interoperation Between the Virtual World Application and Constructive Simulator

In this section, we will introduce the technologies applied during the interoperation between the game application and the constructive simulator. Figure 10 shows the documents used during the *game loop analysis* phase. In order to speed up the pace of development, we utilized PowerPoint documents to determine the data structure between the constructive simulator and the serious game.

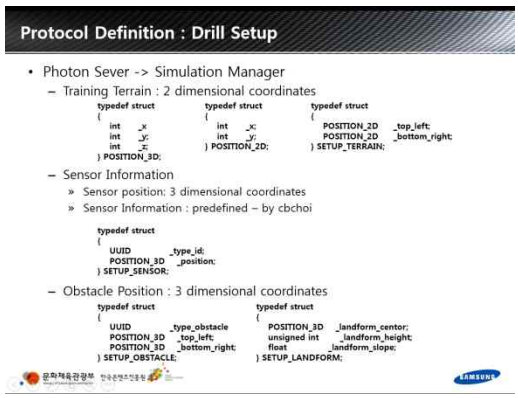

Figure 10: Communication Documents for Drill Setup

Figure 11 shows the calibration concept during the *parameter tuning* phase. The left portion of Figure 11 shows the geographical features that the trainer has arranged. In order to control the path of the evacuation, the trainer may place more virtual objects. The right portion of the figure shows that the constructive simulator has received the geographical features from the serious game. Since every client in the game should receive information about the objects, which are allocated in the virtual space, the SGA receives the information and transfers the data to the HLA/RTI. Then, the constructive simulator receives the information and initializes the geographical features of the field. While transferring the geographical information during interoperation, the SGA discretizes the geographical data spatially.

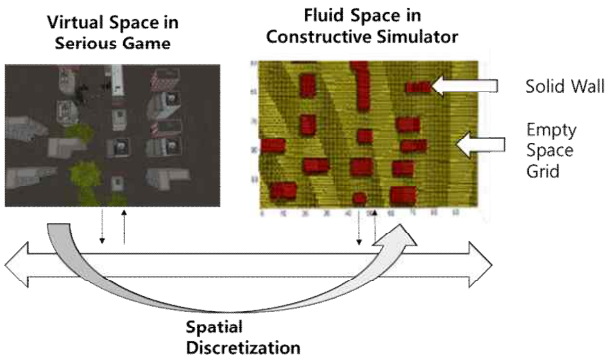

Figure 11: Chemical Compound Simulator

During the training session, the SGA continuously updated the other users' position information to the HLA/RTI, and the constructive simulator assessed live or killed states based on the trainee's chemical dosage amount. After the assessment, the simulator sent the trainee's state to In-World Editor through the HLA/RTI. Finally, we utilized the administrative functionality to make killed trainees lay down.

4.4. Lessons Learned

After developing SGMT using HLA/RTI, we gained several insights. First, depending on the demanded accuracy of the serious game, the constructive simulator can utilize various turbulence CFD models. However, several accurate CFD models cannot guarantee the timing constraints of real-time simulation because of the huge computational time required. Therefore, we had to find appropriate CFD models to satisfy the requirements of a serious game. Moreover, after we found the appropriate CFD models, we had to tune the parameters iteratively until they were appropriate for the CFD model.

Second, in order to develop a federation between the SGMT and the constructive simulator without modifying the SGMT, the protocol between the server and the client of the SGMT should be opened up to the developer. Since we are developing an SGA, which acts as a gateway to the other simulator, the developer should understand the game loop of the serious game. The problem is that commercial games do not offer open game protocols. As a result, we had a difficult time acquiring a serious game in which to develop the federation.

Third, in order to affect the user or the virtual objects during gameplay, a serious game should support administrative features or server-side scripting features. Since the serious game we used was limited for other training contents and we are extending the serious game using HLA/RTI, we could share information easily from the serious game to the constructive simulator. However, if the serious game does not provide the functionality for the user to influence the behavior or states of virtual objects and other users, then it will be very limited in helping trainees to gain training experience. For example, before we discovered In-World Editor's administrative functionality, we displayed the simulation results in the chat area. Because of its functionality, we chose this virtual world application over several other applications. The virtual world application can make up and arrange virtual training fields easily, and supports server-side scripting, so that we can influence the users and the virtual world objects easily.

5. CONCLUSION

Extending a serious game for military training can be tedious and difficult work. In order to support developers in extending serious games more easily, we have proposed a methodology to develop a SGMT using HLA/RTI. The methodology extends the SoSES/FB framework and its development process. The main characteristic of the methodology is that, when a game agent and a constructive simulator are provided, a developer can easily synthesize the federation using the SoSES/FB framework.

In case a game agent or a constructive simulator does not exist, the methodology provides a means to develop a federation. We expect that the proposed MSGDEP will assist developers who want to extend existing game applications to serious games, or extend existing constructive simulators to training simulators.

ACKNOWLEDGMENTS

This work was supported by the Defense Acquisition Program Administration and the Agency for Defense Development under the contract UD110006MD, South Korea.

REFERENCES

SecondLife, 2013. Available from: < http://secondlife.com/ >.

Rippin, P., 2009. Virtual World Simulation Training Prepares Real Guards on the US-Canadian Border: Loyalist College in Second Life. Linden Lab., Available From: <http://secondlifegrid.net.s3.amazonaws.com/docs/Second_Life_Case_Loyalist_EN.pdf>.

Fishwick, P., Kamhawi R. Coffey, A. and Henderson, J. 2010. An Experimental Design and Preliminary Results for a Cultural Training System Simulation: *Proceedings of Winter Simulation Conference* 2010, pp. 799-810, Washington D.C. USA

Virtual Battlespace 2, 2013. Available from: < http://products.bisimulations.com/products/vbs2/>

Maxwell, D., McLennan, K., 2012, June. Case Study: Leveraging Government and Academic Partnerships in MOSES (Military Open Simulator [Virtual World] Enterprise Strategy). In *World Conference on Educational Multimedia, Hypermedia and Telecommunications* (Vol. 2012, No. 1, pp. 1604-1616).

OpenSimulator, 2013. Available from: <www.opensimulator.org>

IEEE1516-2010, 2010. Standard for Modeling and Simulation High Level Architecture – Framework and Rules, 2010.

IEEE1516.3-2003, 2003. IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP), 2003.

Kim, B.S., Choi, C.B., and Kim, T.G., 2013, Multifaceted Modeling and Simulation Framework for System of Systems Using HLA/RTI, 2013 Spring Simulation Multiconference, 16th Communications and Networking Symposium (CNS), San Diego, CA, USA.

Valente, L., Conci, A., and Feijó, B., 2005. Real time game loop models for single-player computer games, In: *Proceedings of the IV Brazilian*

Symposium on Computer Games and Digital Entertainment, pp. 99–107.

Unity 3D, 2013. Available from: <www.unity3d.com>

Photon Network Engine, 2013. Available from: < http://www.exitgames.com/Photon/Unity>

Blazek, Jiri, 2001. *Computational Fluid Dynamics: Principles and Applications: Principles and Applications.* Elsevier

Roe, Philip L., 1981. Approximate Riemann solvers, parameter vectors, and difference schemes, In : Journal of computational physics, 43(2), 357-372.

Park, Sang C., Kwon, Y., Seong, K., and Pyun, J., 2010, Simulation framework for small scale engagement. In *Computer & Industrial Engineering*, 2012 (59), 463–472.

Gwenda, F., 2004, Adapting COTS games for military simulation. In: *Proceedings of the 2004 ACM SIG SIGGRAPH international conference on Virtual Reality continuum and its application in industry* , pp.269–272.