A COLLABORATIVE VIRTUAL TRAINING ARCHITECTURE FOR INVESTIGATING THE AFTERMATH OF VEHICLE ACCIDENTS

Ahmed BinSubaih
Steve Maddock
Daniela Romano
Department of Computer Science
Regent Court, 211 Portobello Street
Sheffield, S1 4DP, UK
Email: a.binsubaih, s.maddock, d.romano@dcs.shef.ac.uk

KEYWORDS
Collaborative virtual training environments, architectures.

ABSTRACT

Recently technologies have improved training practices tremendously, especially practices that require visualization and collaboration to enhance the understanding of any situation and to deploy strategies to solve these problems. Visualization of problems enhances a trainee's understanding of it in a more natural way that mimics everyday practice in the real world. Once a mental picture of the situation has been constructed, the trainee can then try different approaches to remedy it and therefore learn by experience. This paper investigates the use of collaborative virtual environment (CVE) technology for training purposes. We present an architecture for CVEs developed with the purpose of being domain independent by separating scenario logic from the simulation environment. We believe that such separation leads to a flexible environment that can easily be modified into new scenarios. To demonstrate this we have built two scenarios to show how the logic can be separated and controlled from outside the environment. The first scenario allows new police recruits to be embedded in a collaborative virtual environment with other participants (witnesses, operation room operators, etc) with the purpose of investigating and dealing with the aftermath of vehicle accidents. The second scenario allows participants to attend a virtual lecture. In both environments we show how a trainer can monitor the environment and trigger events that unfold the scenario. Placing the logic in the hands of the trainer is our goal to show how a collaborative virtual environment can be made flexible. It also lays the basic ground for a proposed domain-independent architecture.

INTRODUCTION

The main aim of training is to transform the knowledge acquired during the training session into the trainee's collection of recallable experiences. Various technological approaches have been used to enhance this transformation such as: electronic publishing, television and video, teleconferencing, virtual reality, etc. This paper focuses on the use of collaborative virtual environments as a training tool. Virtual Environments (VE) are particularly appropriate for training collaborative emergency skills, as demonstrated by the many successful examples of its use in training; e.g. flight simulators, which have provided invaluable training for pilots. The advantages of virtual environments over other technologies become obvious when examining their main attributes: visualization (Rohrer 2000), presence (Nunez and Blake 2000; Slater and Steed 2000), co-presence (Witmer 1998), and immersion (Casanueva and Blake 2001). These attributes allow a recallable experience to be formed in the trainee's mind (Romano and Brna 2000).

One of the major contributions of this paper is presenting an architecture where the scenario logic is separated from the simulation environment. The aim is to develop a flexible architecture which caters for different scenarios. Moreover, we believe such flexibility confers the following advantages:

- The scenario logic and the simulation environment can be modified independently allowing iterative development cycles and easy substitution of elements in the environment.
- The decoupling of the scenario logic from the simulation environment encourages encapsulation and other good object-oriented coding practices thus easing the generation of CVEs.

The work presented in this paper is both a proof of concept for a domain-independent architecture and the ground work for the generation of the related easy-modifiable collaborative virtual environment. We propose to achieve the domain-independence by introducing an intermediary - called Events Space- between the scenario logic (stored in knowledge base (KB) systems format) and the simulation environment. The advantages of such an intermediary are:

- It enables interoperability between distinct simulation environments, a practice promoted by the High Level Architecture (HLA) (Smith 1998).
- The three different parts (KB, events space, and simulation environment) can be individually

tailored to the expertise and computer literacy of their users (domain experts, trainer, and trainees).

- Automatic generation and control of scenarios for training purposes.
- KB systems can be used to guide trainees during simulation sessions by exploiting their ability to provide explanation of how solutions are derived.

The groundwork in this paper focuses on examining the separation sought on a CVE rather than a VE because of the inherited distributed nature of a CVE and the existence of many different techniques for communication, such as the events mechanisms with its decoupling ability (Eugster et al 2003; Hamman et al 2001).

In order to test our system we have created a scenario to train new police recruits on how to investigate the aftermath of vehicle accidents. The main factor that assisted in choosing this scenario was the accessibility of such information since the project is sponsored by the Dubai Police. Furthermore, the existence of training courses to deal specifically with vehicle accidents allows a comparison between the use of CVE and traditional training courses. The target audience of the training environment is the new police recruits studying at the Dubai Police College and recent graduates working for the Dubai Police Traffic Department and undergoing specialised training.

In this paper, we first describe the related work, followed by a detailed presentation of the proposed architecture. A walkthrough of the scenario creation process is then described. Finally, we illustrate the flexibility of the architecture by deploying a second scenario on it.

BACKGROUND

A number of virtual environments and collaborative virtual environments have been developed over the years with different features examining a wide range of research interests. These environments can be placed in three main categories based on the relationship between the scenario logic and the simulation environment. The three categories are: applications, virtual development environments, and KB systems.

The ACTIVE project (Romano 2001) is an application which is used to train fire fighters. It uses a special technique called ‘super powers’, which give trainees capabilities in a virtual environment that are not available for them in a real life situation, such as rewind, replay or forward the incident to view a demonstration of the solution. CACTUS (Williams 1995) trains police officers to control crowds in domestic incidents. It uses an AI technique to model crowd behaviour so that the crowd can behave autonomously in an environment. Furthermore, it allows the trainer to influence the crowd behaviour and guide the training. The DC-Train (Hamman et al 2001a) system trains Navy personnel on how to control ship damage. This system allows the trainee to receive the status of the ship and send commands to different stations to control and repair the damage before it escalates. All the messages communicated are stored for evaluation and retraining purposes. Finally, CarSim (Akerberg et al 2003) is another system which automatically reconstructs a traffic accident from a textual report. The reconstruction is output in the form of 3D animation.

Reviewing the above applications, we have identified the tight coupling of the scenario logic to the environment simulation as the most restrictive factor that prevents the adaptation of VEs. This inflexibility has a negative impact on the use of the virtual environment for training purposes since it leads to simulation memorization where the trainee starts to memorize the events sequence and understands how to ‘beat’ the simulation. The other symptom shown by many current environments, which can be attributed again to the inflexibility issue, is that they tend to be domain dependent. This makes it impossible to transfer the training feature of a system from one application to another without substantial work.

The second category contains virtual development environments (Hawkes and Wray 1998; Tamberend 2003; Cruz-Neira et al 2002; Singhal and Zyda 1999; Shaw et al 1992; Wang et al 1995) that are built specifically for the use of developers producing virtual environments. These environments normally ease the development lifecycle by abstracting the low level complexities such as interacting with VR devices. However, some of these development environments are no different than using a programming language in the sense that it places the creation of the link between the logic and the simulation environment in the hands of the developer. Nevertheless, some of these development environments (Hawkes and Wray 1998; Tamberend 2003) encourage flexibility by providing a high level scripting language, thus making it feasible to separate and modify the scenario logic. The Environment Manager (EM) tool (Wang et al 1995) even provides a higher level of support in the form of a simple script file for creating environments.

Finally, KB systems (Tecuci 1998; Szarowicz et al 2002; Gonzalez and Douglas 1993) are geared towards separating the logic or knowledge from the system using it. They use an inference engine to deal with retrieving the appropriate results. The other advantage described in (Gonzalez and Douglas 1993) is the ability to make the knowledge domain-specific, which means that it can accommodate different domains by simply changing the knowledge. Furthermore, the separation of the logic from the system also allows the modification of the knowledge independently from the simulation environment and more frequently without the developer involvement, which means there is no need for recompiling the simulation.

Summarising, the drawbacks of some of the previous categories are:

- Embedding the logic in the simulation environment makes it inflexible to change.
- Even in KB systems the logic usually tends to be specific to the simulation environment considered and requires some work to be able to reuse it in a

different simulation environment. This usually makes the logic created limited to a specific simulation environment.

Some of the strengths of the previous categories are:

- Interoperability between different simulation environments.
- The separation of the logic from the simulation environment shown by the KB systems.
- The decoupling accomplished using events mechanisms.
- Providing a high-level scripting language makes the interaction with the simulation environment less complex.

Our work proposes a twofold solution to develop an architecture for domain-independent collaborative virtual environment. The first phase, illustrated in this paper, focuses on investigating the feasibility of developing an architecture that targets the separation of the scenario logic from the simulation environment using it. The second phase examines the challenges of adopting a domain-independent approach by introducing an intermediary between the domain knowledge and the simulation environment. The role of the intermediary is to automatically generate and control scenarios using domain knowledge elicited from experts.

THE SYSTEM

The main aim of the system presented here is to address the flexibility issue in collaborative virtual environments. Two types of users are going to interact with the system: trainers and trainees. Therefore, the system should allow participants to co-exist in a virtual environment and be able to communicate. The following sections describe the requirements and the implementation of the system.

Requirements

System requirements are divided into three sections: trainer requirements, the requirements of the players, and scenario requirements. The trainer should be able to carry out the following tasks:

- Create scenario behaviour: allows the trainer to create different scenarios by adding events, e.g. specifying paths for objects to follow or attaching sounds to objects.
- Monitor scenario behaviour: permits the trainer to watch the scenario unfolding by joining the scenario as an invisible participant.
- Control scenario behaviour: allows the trainer to stop or change the course of the simulation behaviour to fit the training requirements.
- Insert a scene layout: permits the trainer to insert different scenes making the architecture scene independent.

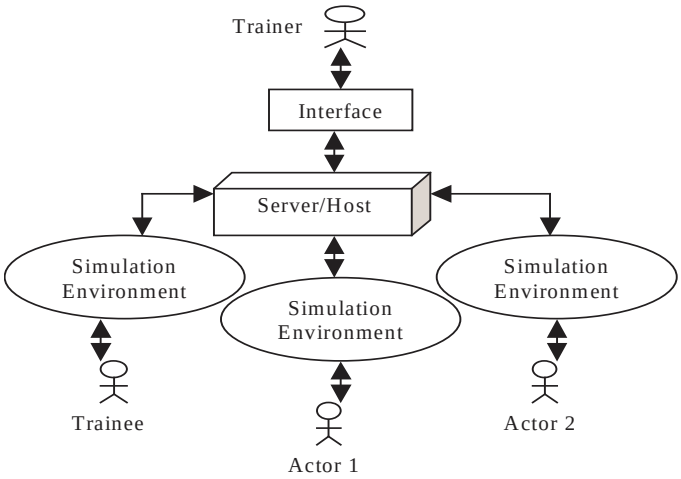

Figure 1. System architecture

The players should be able to join the environment and communicate with other players using voice communication. In addition , to enable a better perception of the location of other players in the environment, a positional sound feature should be added to the voice to allow players to locate each other. Text communication is not used because the delay caused (see the 'dead moment' described in (Slater and Steed 2000)). Moreover, typing also means the player breaks his presence as he is required to achieve the communication task in an unnatural way (using the keyboard to type the message).

The vehicle accident scenario aims to train new traffic officers on how to deal with the aftermath of vehicle accidents and how to investigate them. The officer is placed in a virtual environment where he uses an input device move around and a headphone/microphone to communicate with other participants in the same environment, but who might be in different geographical locations. His role is to collaborate with the participants to resolve the matter and find out how the accident occurred by questioning the participants and examining cues in the scene (e.g. damages to vehicles, skid marks, etc). The trainer should also be able to join the same environment, as an invisible participant, monitor the scenario unfolding and trigger events (e.g. ambulance animation, siren, etc).

Implementation

This section details the system architecture, simulation environment design, network topology, events model, and object model.

Conceptual System Architecture

Figure 1 shows the system architecture. It is based on the client/server network topology. The role of the server is to host the session and forward the communication between the participants. It also allows the trainer to carry out the tasks described earlier.

Table 1. Scenario behaviour commands

Behaviour	Command
Enable/Disable collision detection	SetCollision(ID,flag)
Enable/Disable object visibility	SetVisiblity(ID,flag)
Set object position and orientation	SetPlayerPosition(ID,x,y,z,xRot,yRot,zRot)
Play the animation attached to skinned mesh	SetPlayerAnimation(ID,flag,name)
Set a sound source and attach it to object in the scene	Set3DSound(ID,file)
Play a sound source	PlaySound(ID)
Set the sound minimum and maximum distances	SetSoundDistance(ID,min,max)
Show/Hide object bounding box	SetRenderingBoundingBox(ID,flag)
Add animation route based on four points and give it name, duration and loop flag	AddRouteAnimation(ID,p1x,p1y,p1z,p2x,p2y,p2z,p3x,p3y,p3z, p4x,p4y,p4z,duration,loop,name)
Play one of the animation routes attached to the object	SetRouteAnimation(ID,loop,playing,name)

Simulation Environment Design

Figure 2 shows the simulation environment and how a high level scripting language is embedded across three architectural layers (DirectX 9.0, game engine, and simulation environment) to enable the manipulation of the scenario behaviour. The scripting language used is Python which allows easy replication of the game engine classes that are based on an object-oriented approach using C++.

Furthermore, the other advantage scripting provides to this prototype is the ability to dynamically load code at run-time which can be used to insert and control the behaviour. Figure 2 shows how a game engine has been built on top of DirectX to abstract all of its complexities. The layer above the game engine is the simulation environment, which holds the objects, the players, and the scene settings. The top layer is the behaviour controller which allows for simple run-time access to the environment to allow the trainer to broadcast events to insert or change the behaviour of any objects, players, or part of the environment. The different set of behaviours that are allowed to be inserted and manipulated during run-time are shown in Table 1.

Network Topology

The client/server approach is used in this prototype, and it is achieved by using DirectPlay, a component of DirectX 9.0. DirectPlay allows a session to be initiated for players to join and communicate. Each player, after joining the session, gets a unique identification number. Players communicate with each other through the server holding the session by sending messages to specific players or broadcasting to all players.

The list of all available players can be queried from the server. As players join or exit the session, the server sends special messages to all the other players notifying them of the player's action. The same approach is used to allow players to interact and exchange information to comply with the co-presence requirement mentioned earlier.

The voice session uses the same DirectPlay session to communicate voice packets between players and the server. Voice has three different communication topologies: forwarding server, mixing server, and peer-to-peer. The forwarding server forwards the voice packets to the clients who decompress the packets and process them to emulate, for instance, positional voice. The mixing server, on the other hand, mixes the voice packets coming for all players and then forwards them to the players. This approach does not allow for positional support because the players only receive the mixed packets of the voice. The third type supports positional voice but cannot be used with the client/server approach adopted in the prototype described in this paper. The prototype developed uses the forwarding topology because of its support for 3D and the client/server approach.

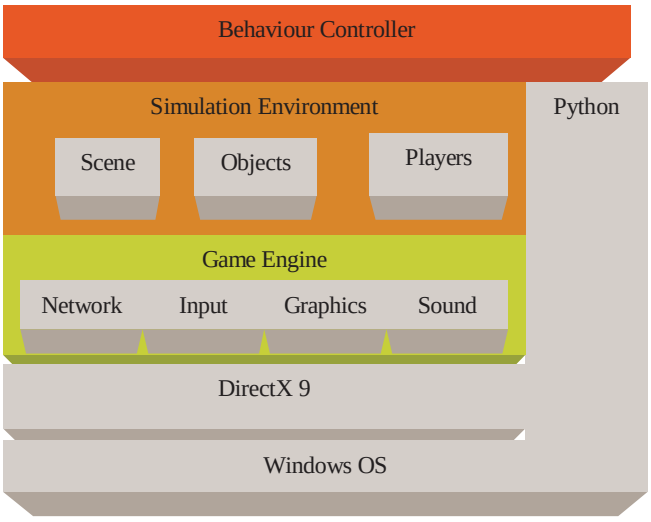

Figure 2. Simulation environment design

Object Model

The next element in addressing the system design is the management of the dynamic shared state of the simulation to enable the synchronization of game status among all participants. This is achieved by sharing the object model, which comprised of the scene object, the player object, and the environment settings. The scene object and player object share common properties such as position, orientation, mesh, animation, sound, etc. A parent object

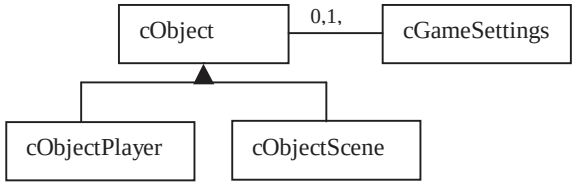

Figure 3: Object Model

Table 2: The messages flow between the client and server for joining and initializing a player (@ is a separator)

Seq	Direction	Message	Format
1	Server	Player joined	DPNMSG_CREATE_PLAYER
2	Client	Joining process complete	DPN_MSGID_CONNECT_COMPLETE
3	Server-to-Client	Groups	100@dpnidplayerID@ID@bMonitor@bModeration@bInvisible@bForcelookAt@bDriver@bCRWnd@bCRRadio@bChatWnd@bAvatar@title@tasks
4	Server-to-Client	Meshes	101@dpnidplayerID@ID@Name@Desc@File@Picture
5	Server-to-Client	Players	102@dpnidplayerID@m_iPlayerID@m_iGroupId@m_iMeshID@m_iVehicleBeforeID@m_iVehicleAfterID@m_iMotionID@takenStatus@m_strTitle

has been created to hold all the common attributes (cObject). The player object (cObjectPlayer) and the scene object (cObjectScene) inherit the class properties from 'cObject' as shown in Figure 3. The scene settings are stored in the cGameSettings class. The way these are synchronized across the participants is described in the next section.

Event Model (State synchronization)

The synchronization of the virtual environment between all participants is accomplished in two phases. The first phase sends the initial information to the participant at the time of joining the simulation session. This information includes current scene objects with their attributes such as position, orientation, mesh, visibility, etc. The players information, however, is sent as they join the game along with their attributes, except for those players who are currently in the session whose information gets sent after the initial information.

Table 2 shows the player initialization messages sent between the client joining the simulation, the server, and the other clients who are already in the game. It also shows the direction, and the message format used. DirectPlay triggers the first message on the server after receiving a request by a client to join using the IDirectPlay8Client::Connect method. This call triggers the following message on the server: DPN_MSGID_CREATE_PLAYER. The message contains a unique identifier (dpnidPlayer) variable assigned to a player after joining.

Once the connection attempt is completed, a second message is created by DirectPlay and sent to the client - DPN_MSGID_CONNECT_COMPLETE. The message contains the unique identifier (dpnidLocal) given to the player which is used to identify him during any future communication. Players can be organized in groups that describe common features such as a different type of user interface. For example, an operation room operator might not require a 3D interface to see the environment as he communicates using voice only. The messages are identified by a manually created identifier. The client then chooses a player from the unallocated players and sends a message to the server and the allocation result is sent back to the client. This continues until a player is allocated to the client. Then the client is allowed to enter the simulation.

After the client has joined the game, he communicates with the server and other clients using the messages shown in Table 3. These messages allow clients to synchronize positions, orientation, animation, etc. For example, a message with ID 4 sends the position and orientation accompanied by the dpnidPlayer which uniquely identify each player and allow the clients receiving the message to update the appropriate avatar.

Finally, the player can exit the session by requesting it from the server or by the server terminating the session without request using IDirectPlay8Client::Close() and sending a DPN_MSGID_TERMINATE_SESSION message respectively. When the player leaves the simulation other players are informed by receiving a DPN_MSGID_DESTROY_PLAYER message which has the dpnidPlayer number.

SCENARIO IMPLEMENTATION

This section details the vehicle accidents scenario implemented on the environment. The scenario has been developed specifically to show how the logic can be separated from the simulation environment and how it can be manipulated offline and online during the simulation run. The scenario creation process is also described along with how the environment allows the trainer to act as the 'brain' of the simulation by monitoring the environment and deciding the course of actions.

Vehicle Accidents Scenario

The scenario chosen allows five participants to join the simulation: traffic officer, two drivers, operation room operator, and trainer. The officer and the two drivers are represented by avatars in the virtual environment, whereas, the operator has a voice communication only and no physical representation in the environment (i.e. no avatar). Each avatar has some behaviours attached to it such as movement and orientation to allow the avatar to traverse the environment. The trainer can also join the environment as an invisible participant to monitor and control the scenario.

The vehicle accident scene contains two vehicles involved in an accident, and one injured passenger. The scene also

Table 3. The messages flow between the clients and the server during the simulation session (@ is a separator)

ID	Direction	Message	Format
4	Client-to-Server-to-All Clients	Position and orientation	4@dpnidPlayer@ID@x@y@z@xRot@yRot@zRot
444	Client-to-Server-to-All Clients	X Animations statuse	444@dpnidPlayer@GameID@Count@AnimationsTex t
5	Client-to-Server-to-All Clients	Chat message	5@dpnidPlayer@ID@chatText
900	Client-to-Server-to-All Clients	Set 3D Sound	900@dpnidPlayer
901	Client-to-Server-to-All Clients	Python script	901@script

has other objects such as roads, buildings, fences, other vehicles, etc. The injured passenger behaviour can be in the form of a screaming sound which is triggered at certain time by the trainer and possibly stopped when the ambulance takes him away from the scene.

The simulation starts with a call from the police operation room informing the trainee of an accident and requesting him to attend. The trainee then finds himself at the scene of the accident where he can navigate the environment. The trainee's objectives are to bring the situation under control, investigate the incident, and produce an accident report. He can navigate around the scene and communicate with the operation room to request additional resources as needed.

Once the resources are deployed (simulated or played by other actors) and the injured passenger is then taken to the hospital, the trainee can start investigating what happens by examining the scene's clues and by interviewing the two drivers and any witnesses. The interview is conducted using positional voice communication between the participants.

A sample script of a possible scenario is shown in Figure 4. Such a script is usually designed by the scenario creator and hard-coded into the simulation environment or scripted in some way. In our current implementation of the architecture, rather than hard-coding this logic it is left to the trainer to trigger the appropriate events at appropriate times. For example, the trainer might start the injured person screaming sound at the third minute during the simulation. This might be followed by the pain animation at the fifth minute, and then the trainer will wait for the trainee's actions to decide on what to do next. For example, if the trainee calls an ambulance by contacting the operation room by voice, the trainer can then launch the ambulance animations.

Scenario Creation Lifecycle

The scenario creation passes through the following steps: the creation of the models of the objects to be used in the environment, the scene layout setup, and scenario logic creation.

The models creation is carried out using a modeling tool, e.g. 3D Studio Max 6.0. Once a model is created and optimized it is exported to the X file format which is then used by the Scenario Creator, as shown in Figure 5, to store the objects in the database to make them reusable for different scenes. The database contains five tables, three of which are specific to the scenario (games, players and scene objects) and the other two hold reusable information (meshes and groups).

The second step sets the scene layout by positioning and orienting the scene objects. The Scenario Creator allows the insertion and positioning of the objects in the environment. It also allows assigning a title to the object to ease its identification. Each object created has its own unique identification number. The number of players allowed in the environment and the avatars attached to them is set in the scenario configuration step using the players tab in the Scenario Creator tool. The scene creation process is shown in Figure 6.

Once the scene layout is ready the next step is to create the scenario logic, an example of which is shown Figure 4. This involves deciding on the path of animations and the timing of events. This is achieved by directly manipulating the functionality of the simulation environment through the embedded high level scripting language (Python) which allows run-time loading of behavioural scripts. Table 1 shows samples of the behaviours available to the trainer. For instance, the screaming behaviour is added using the 3D sound insertion method. The method takes an object id and a sound source file and attaches the sound to that object using the following method: 'Set3DSound(ID,file)'. The scenario controller can then play the sound at any time using 'PlaySound(ID)' which takes an id of the object on which the sound has been attached. He can also stop it by using 'StopSound(ID)'. Furthermore, he can set the distance at which the sound can be heard using 'SetSoundDistance(ID,min,max)'

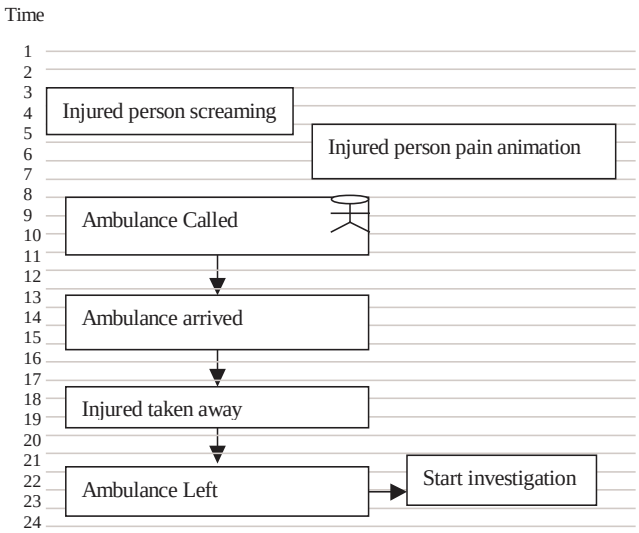

Figure 4. Sample scenario

which takes the object id and the minimum and maximum distances. The minimum distance is the distance at which the sound is uniform, whereas the maximum is the distance after which the sound cannot be heard. The sound decreases when moving from the minimum to the maximum distances. The scripting language demonstrated in this example could be easily hidden in a final application by implementing a graphical interface to hide the methods. This would allow the logic to of the environments to be designed by computer users at any level.

Running the Scenario

Table 4 shows the steps taken by the trainer in running and controlling the scenario. Once all the players have joined the scenario the trainer can start broadcasting the scenario behaviours. To run the scenario shown in Figure 4, a trainer joins the simulation and acts, for example, as the operation room controller. The purpose of him joining the simulation is to trigger events based on the conversation between the players. For instance, if the trainee calls an ambulance the trainer triggers the ambulance event. The trainer also monitors the virtual environment and carries out all the manual tasks (e.g. take injured passenger away). Currently all actions are broadcasted as Python scripts, which are interpreted and acted on by each simulation environment.

RESULTS

During the initial informal experimental run of the scenario, the architecture was tested by allowing four players to join the environment (traffic officer, two drivers and the operation room operator). The trainee played the role of the traffic officer. The drivers were played by two actors who were given different scripts of their details, and a short description of how the accident occurred from their point of view. The operator was played by the trainer who also controlled the scenario by broadcasting events. The trainer used the scenario script shown in Figure 4 to broadcast the appropriate events at the specified time. Figure 7 shows the scenario running on a wide-screen Reflex setup (some of the models have been downloaded freely from turposquid.com and 3dcafe.com). Figure 8a displays the scene from the perspective of one of the player.

The second scenario run on th system (virtual lecturing) is shown in Figure 8b. This scenario was created to demonstrate the flexibility of the proposed approach and the strengths of the architecture. In the virtual lecturing scenario, four participants join the same environment to attend a lecture given by a fifth participant. The same process of creating the scenario was undertaken but with a reduced time as the architecture allows reusability of the general data.

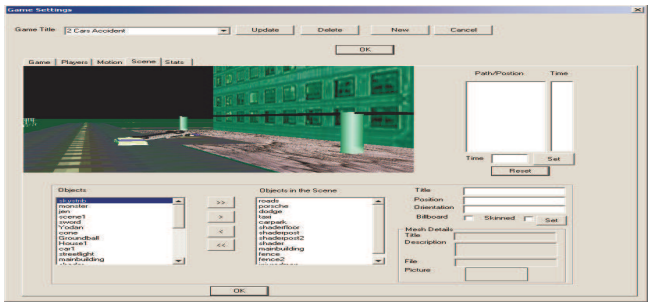

Figure 5. Scenario Creator

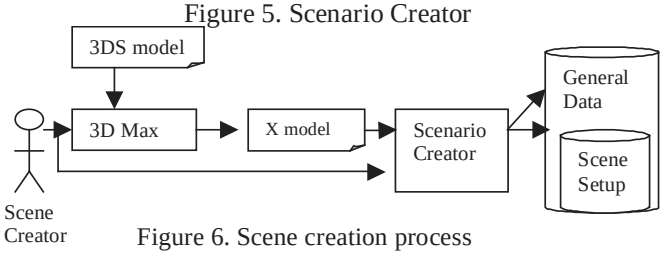

Figure 6. Scene creation process

Figure 7. Accident scenario running on Reflex Studio

(a)

(b)

Figure 8. Vehicle accidents (a), virtual lecturing (b)

Table 4: Scenario controller or trainer tasks

Order	Task
1	Choose a scenario from the drop down list
2	Choose the network card for the server session
3	Start the voice communication server
4	Wait for players to join
5	Start broadcasting behaviours

CONCLUSIONS

The separation of the domain knowledge and the ability to control it at run-time were the two primary objectives of the current prototype. The separation has been achieved by the use of a high level language to create a layer on top of the simulation environment to act as an interface. This meant that behaviours could be inserted and controlled at run-time. The abstraction achieved by the scripting language made controlling the application behaviour much easier and dynamically loadable at run-time. Moreover, using an existing language (Python) rather than building our own reduced the development time as it eliminated the need to build a parser and evaluate it. The other advantage of this approach is that it allows commands to be tailored to suite the user's computer literacy, as another simplification layer can be added such as graphical user interface.

By creating and running two different scenarios on the prototype simulation environment we have demonstrated the flexibility of the approach since it managed to detach the domain knowledge from the simulation environment and allow its control at run-time. These are promising results. However, to prove the architecture flexibility more tests will be carried out involving scenarios from different domains. The observation that can be made about the flexibility of the prototype is that the more the simulation environment functionality is exposed through the scripting language the more flexibility is achieved. This means that if the full simulation environment functionality is exposed, then the system can be labeled fully flexible. We define flexibility as the ability to insert and control behaviours at run-time.

Also we have shown that the current system can handle running multiple environments while allowing a trainer to monitor the simulation and trigger events to create the desired scenario.

In the next stage of development we aim to replace the trainer by the architecture proposed in Figure 9. This should be compared with Figure 1 to see that a human trainer with expert knowledge will be replaced by a KB and an events space. In this proposed architecture we place an intermediary between the scenario logic and the simulation environment in an attempt to be domain and simulation environment independent simultaneously.

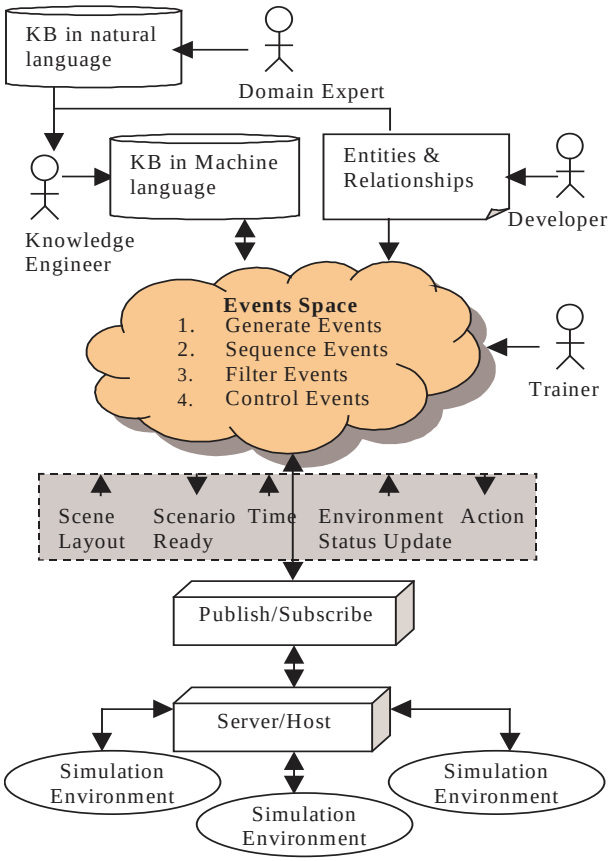

Figure 9. Events space as a link between knowledge base and simulation environments

ACKNOWLEDGMENTS

This project is sponsored by Dubai Police. Thanks to James Edge for his review of the paper. Also thanks to Michael Meredith (who administers the Reflex Studio) for his support and suggestions that enabled the tool to run on the studio set-up.

REFERENCES

Akerberg, O., Svensson, H., Schulz, B., Nugues, P. "CarSim: An Automatic 3D Text-to-Scene Conversion System Applied to Road Accident Reports". Research Notes and Demonstrations Conference Companion, EACL 2003: 191-194.

Casanueva, J. and Blake, E. "The Effects of Avatars on Co-presence in a Collaborative Virtual Environment". Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT2001). Pretoria, South Africa. September 2001.

Cruz-Neira, C., Bierbaum, A., Hartling, P., Meinert, K., Just, C. "VR Juggler – An Open Source Platform for Virtual Reality Applications". AIAA 2002 Aerospace Science Conference, Reno, NV, January 2002.

Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A. "The many faces of publish/subscribe". In ACM Computing Surveys (CSUR), volume 35, issue 2 (June 2003), pages: 114 - 131

Gonzalez, A., J., Douglas, D., D. "The Engineering of Knowledge-Based Systems, Theory And Practice". ISBN 0-13-334293-X, Prentice Hall International Editions, 1993.

Hamman, M., LeMentec, J. C., Wilkins, D. C., "Design Requirements for DC-Train 4.0". Knowledge Systems Lab Report UIUC-BI-KBS-2001-0029. Beckman Institute, University of Illinois, Urbana-Champaign. February 2001.

Hamman, M., Wilkins, D. C., Carbonari, R., Mueller, C. "DC-Train 4.0 Instructor's Manual". Knowledge Systems Lab Report UIUC-BI-KBS-2001-0040. Beckman Institute, University of Illinois, Urbana-Champaign. November, 2001.

Hawkes, R., Wray, M. “LivingSpace: A Living Worlds Implementation using an Event-based Architecture”. HPL-98-181, Extended Enterprise Laboratory, 1998.

Nunez, D and Blake, E.H.. “Cognitive Presence as a unified concept of virtual reality effectiveness”. UCT Technical Report CS01-11-00.

Rohrer, M. W. “Seeing Is Believing: The Importance of Visualization In Manufacturing Simulation”. Proceedings of the 32nd conference on Winter simulation Orlando, Florida, pages: 1211 – 1216, 2000, ISBN:1-23456-789-0

Romano, D. “Features that Enhance the Learning of Collaborative Decision Making Skills under Stress in Virtual Dynamic Environments”. Ph.D.thesis, Computer Based Learning, University of Leeds, UK, August 2001.

Romano, D.M and Brna, P “Presence and Reflectioning Training: Support for learning to Improve Quality Decision Making Skills under Time Limitations”. CyberPsychology & Behaviou Mary Ann Liebert Inc., 4:2, pp265-277

Shaw, C. Liang, J, Green, M. Sun, Y. “The Decoupled Simulation Model for Virtual Reality Systems”. In Human Factors in Computing Systems CHI'92 Conference Proceedings, pages 321-328, Monterey, California, May 1992. ACM SIGCHI.

Singhal, S., Zyda, M. “Networked Virtual Environments Design and Implementation”. ISBN 0-201-32557-8. ACM Press, 1999.

Slater, M. and Steed, A. “A Virtual Presence Counter”. Presence: Teleoperators and Virtual Environments 9(5), 413-434, 2000.

Smith, R. “Essential techniques for military modeling and simulation”. Proceedings of the 30th conference on winter simulation, 1998, pages: 805 - 812 ISBN:0-7803-5134-7

Szarowicz, A., Forte, P., Amiguet-Vercher, J., Gelepithis, P. “Application of Autonomous Agents for Crowd Scene Generation”. 2nd Hellenic Conference on AI SETN-02, vol. 2 April 11-12, Thessaloniki, Greece, 2002

Tamberend, H. “Avocado: A Distributed Virtual Environment Framework”. Ph.D.thesis, University of Bielefeld, 2003.

Tecuci, G. “Building Intelligent Agents”. Academic Press, 1998, ISBN: 0-12-685125-5.

Wang, Q. Green, M, Shaw, C. “EM – An Environment Manager for Building Networked Virtual Environments”. IEEE Virtual Reality Annual International Symposium (VRAIS 95), pages 11-18, Research Triangle Park, North Carolina, March 11-15, 1995, IEEE.

Williams, R, J. “A Simulation Environment to Support Training for Large Scale Command and Control Tasks”. Ph.D. thesis, School of Computer Studies, University of Leeds, UK, December 1995.

Wright, I. P. and Marshall, J. A. R. “RC++: a rule-based language for game AI”. In: Proceedings of the First International Conference on Intelligent Games and Simulation (GAME-ON 2000). SCS Europe BVBA

Witmer, B. G. and Singer, M. J. “Measuring presence in virtual environments: A presence questionnaire”. Presence: Teleoperators and Virtual Environments, 7(3), 225-24