A Domain-Independent Multiplayer Architecture for Training

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

***Abstract*- In this paper the two issues we address in developing an architecture for training are flexibility and ease of scenario generation. Flexibility is the need to make the architecture domain independent by investigating how the logic (dictating the scenario behaviour) is linked to the simulation environment (where the rendering, networking, etc., occurs). For the second issue we consider how to automatically generate training scenarios from expert systems (ES).**

Our architecture shows how an intermediary "events space" can be used to achieve flexibility by separating the logic from the simulation environment. To show the training suitability of the architecture we have developed a scenario on top of a prototype of the architecture to train new police recruits how to investigate vehicle accidents.

I. INTRODUCTION

A number of virtual environments (VEs) and collaborative virtual environments (CVEs) have been developed over the years, looking at such things as training firefighters [13], police officers [20] and navy personnel [10]. The main issue with such systems is that they are domain dependent, which makes it difficult to reuse their simulation systems on different domains without extensive work, which is expensive and time-consuming. As identified in [5] domain specificity in system development leads to highly inflexible applications. In this paper we propose an architecture that is domain independent.

Many similarities exist between collaborative virtual environments and multiplayer games such as co-existence of players, game state replication, communication protocols, etc. We investigate the similarities in architectural needs from the reusability point of view. More specifically we examine the reasons behind the lack of reusability in collaborative virtual environments architectures and the suitability of adopting some of the approaches from the games world to address them. Thus we have developed a prototype architecture which makes use of the successful games approach of embedding a high-level scripting language.

Furthermore, to widen the use of the architecture for training we propose a technique for automatically generating scenarios, which uses ES and more specifically knowledge-based systems (KB) which inherently promotes the separation of the logic from the simulation environment.

In the next section we discuss related work drawing out some of the problems with current architectures. We then present our full architecture, before presenting some results of using a prototype version for traffic accident training for police officers in the Dubai police force. We conclude with some general comments about the potential of our architecture.

II. RELATED WORK

In this section we describe the architectures used to structure simulation environments and categorize them with regards to the relationship between them and the scenario logic. Our categorization examines the relationship in terms of the communication approaches employed to control the scenario logic and the flow of the scenario logic from its source to the simulation environment. We also describe the personnel involved along the flow path. By describing the personnel we try to relate what we have already implemented with the proposed architecture, since our initial prototype shows the trainer inserting and controlling the scenario logic to distributed virtual environments.

With regards to the personnel involved in creating the scenario, we list four of them: domain experts, scenario creators/trainers, animators and developers. The domain experts provide the expertise of how the simulation should operate by describing the sequential and reactive events. This knowledge is then passed to the scenario creator, who is familiar with the capabilities of the simulation environments. The scenario creator either inserts the knowledge himself or passes it to the developer. The developer is the one responsible for programming the logic in the simulation environment. The less the developer is involved in the cycle of creating or modifying scenarios the more flexible the system is. The animator designs and animates the objects of the virtual environment.

We can consider the following different categories based on the relationship between the scenario logic and the simulation environment: applications, virtual development

environments, commercial software environments, and KB systems.

Certain applications [13] tend to embed the scenario logic inside the simulation environment code and most likely changes to the logic require recompiling the simulation system. The personnel involved will always require the developers to change the scenario because of the application inflexibility. Other applications [1,9,20] attempt to give some ability to modify the logic without recompiling, thus eliminating the developers involvement in the scenario creation cycle. This is accomplished by providing ways for the domain experts or trainers to insert the logic into the simulation environment. However, these applications are usually very domain dependent.

Some virtual development environments [4,11,14,18,19] 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 [11,18] encourage flexibility by providing a high level scripting language, thus making it feasible to be used for separating and modifying the scenario logic. The personnel involved are the same as the ones described for applications above since the developers have control over how the applications, built using these development environments, operate. One tool [19] even provides a higher level of support in the form of a simple script file for creating environments.

Commercial software environments (such as DI-Guy™[1] and Vega™[2]) address domain independence in a much better way by allowing the logic to be inserted using a graphical interface or a high-level scripting language. These succeed in achieving domain independence but usually the logic is in a proprietary format, thus lacking the simulation environment independence. One of the strengths of these tools is their ability to cater for a wide range of users by providing interaction methods of different levels of complexity. Such tools can be used by domain experts or scenario creators using the graphical interfaces provided. The developers also can make use of the API access provided. Games engines also cater for multiple scenarios and scenes, but again the logic is formatted in a proprietary format.

KB systems [10,17] are geared towards separating the logic or knowledge from the system using it. They have an inference engine to deal with retrieving the appropriate results. Furthermore, the separation also allows the modification of the knowledge independently from the simulation environment and more frequently without developer involvement, which means there is no need for recompilation of the simulation.

The drawbacks of some of the previous categories are:

- The embedding of the logic in the simulation environment makes it inflexible to change.
- The logic usually tends to be specific to the simulation environment 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.
- Providing a high-level scripting language makes the interaction with the simulation environment less complex.

In comparison with the above categorization we propose a new category we call ‘simulation services’ (similar to web services). This category attempts to combine the best practices from the above categories and also avoid their drawbacks. The main goal of this category is to make the simulation ‘brain’ run as a service by linking it with the logic from one side and the simulation environment from the other side, therefore advocating independence from both sides. The independence from the simulation environment means that the ‘brain’ can be reused to service other simulation environments, possibly built using different languages, as long as they conform to common communication protocols.

III. ARCHITECUTRE

The simulation environment architecture we pursue puts an intermediary between the scenario logic and the simulation environment in an attempt to be domain independent as shown in Fig.1.

The distinguishing factor of the proposed intermediary lies in the way it is modularised and run as a separate service provider, used to automatically generate and service scenario behaviours to a simulation environment. The scenario behaviours are constructed from a knowledge base representing a specific domain. Participants in the simulation environment communicate their status to the intermediary and receive events to decide their course of action. The

[1] http://www.bdi.com/

[2] http://www.multigen-paradigm.com

communication between the simulation and the intermediary is achieved through the use of an events service and more specifically the publish/subscribe mechanism.

We suggest that using a KB system from one side and the publish/subscribe event mechanism on the other side will not only achieve the proposed separation goal but it will also contribute immensely to making the intermediary module, the *events space*, fully decoupled from the simulation environment side.

In Fig.1. the directed arrows show the flow of information. The events *space* uses four inputs to link the KB to the simulation environments in the proposed architecture:

- Entities and relationships: holds a description of how events can take place in the simulation environment (i.e. the capabilities of the simulation environment). These capabilities at low level reflect the functionality of the simulation engine that can be accessed by an outside party (the *events space*).
- KB: holds the knowledge of the considered domain acquired from the domain expert by the knowledge engineer. In its natural language form, it can be represented using simple rules in the form of 'IF (condition) THEN (action)'.
- Simulation environment: from which the *events space* requires three types of information: the scene layout, the simulation time, and changes in the environment status. The information that the *events space* sends is an acknowledgement of the readiness of the scenario and messages describing what should happen next in the simulation environment based on the current time and environment status.
- A Trainer: interacts with the *events space* in three ways: he can manually create and modify scenarios, he can filter automatically created scenarios, and he can monitor currently running scenarios to redirect the training course to guide the training along different tracks.

IV. Automatic Scenario Generation

The process of creating scenarios starts by creating single events from the knowledge stored in the KB. These events are filtered into plausible and implausible events based either on human intervention or set criteria (e.g. event duration). The filtered events are then sequenced to make up the scenario. These sequences are passed again through a filter to validate them. When the simulation environment passes its scene layout to the *events space*, a sequence of possible

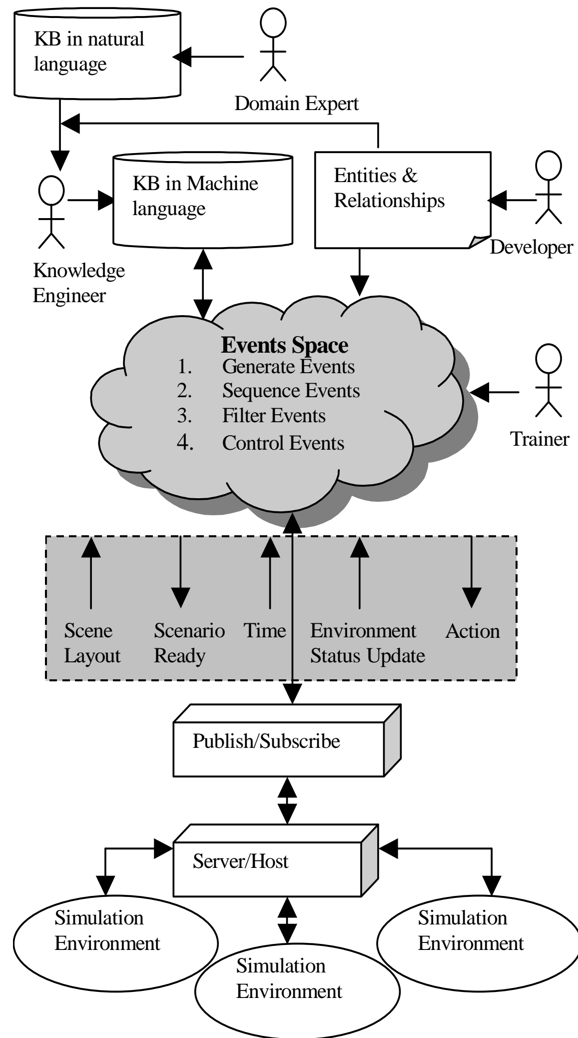

Fig.1. Events space as a link between knowledge base and simulation environments

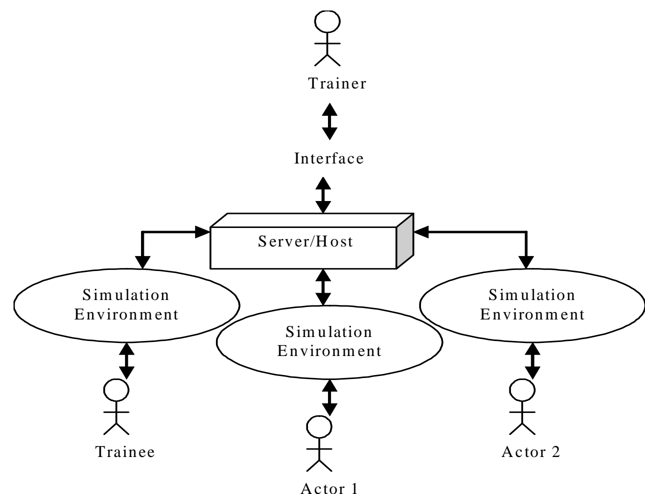

Fig.2. Prototype architecture

Table 1: Scenario behaviour commands

Behaviour	Command
Enable/Disable collision detection	p.SetCollision(ID,flag)
Enable/Disable object visibility	p.SetVisiblity(ID,flag)
Set object position and orientation	p.SetPlayerPosition(ID,x,y,z,xRot,yRot,zRot)
Play the animation attached to skinned mesh	p.SetPlayerAnimation(ID,flag,name)
Set a sound source and attach it to object in the scene	p.Set3DSound(ID,file)
Play a sound source	p.PlaySound(ID)
Stop a sound source	p.StopSound(ID)
Set the sound minimum and maximum distances	p.SetSoundDistance(ID,min,max)
Show/Hide object bounding box	p.SetRenderingBoundingBox(ID,flag)
Add animation route based on four points and give it name, duration and loop flag	p.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 using the above function.	p.SetRouteAnimation(ID,loop,playing,name)

events from the accepted scenarios is chosen. This sequence must conform to the content of the environment layout. For example, no animation of person in pain would be shown unless there was an injured person in the scene layout provided. Automatic scenario generation has not yet been implemented in the prototype described next.

V. PROTOTYPE

The main aim of the system presented here is to address the flexibility issue in multiplayer architectures. Three types of users are going to interact with the system: trainers, trainees, and actors to help with the scenario. Therefore, the system should allow participants to co-exist in a virtual environment and be able to communicate. The following sections describe the system and its implementation.

A. *System Overview*

Fig. 2 shows the developed prototype architecture which allows multiple players to co-exist and communicate using voice. The system permits the trainer to carry out the following tasks:

- Create scenario behaviour: allows the trainer to create different scenarios by specifying the number of participants and their properties. The trainer can also add 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.

The players are able to join the environment and communicate with other players using positional voice

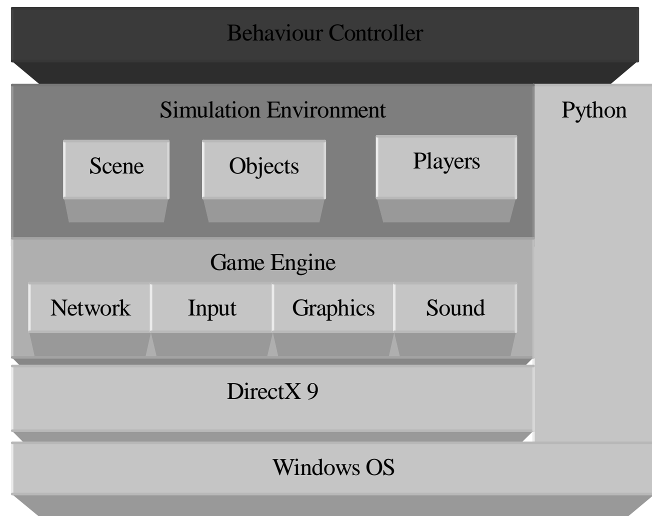

Fig.3. Simulation environment design

communication, which helps players locate each other. Text communication is not used because of the delay caused

(see the 'dead moment' described in [15]). 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 implemented on the prototype 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 (joystick or mouse and keyboard) to move around and an audio headset 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. damage to vehicles, skid marks, vehicle positions, etc).

B. *Implementation*

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

Fig. 3 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++.

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 behaviour. 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.

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.

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 in [2].

The voice session uses the same DirectPlay session to communicate voice packets between players and the server. The prototype developed uses the forwarding topology because of its support for 3D and the client/server approach.

The management of the dynamic shared state of the simulation to enable the synchronization of game status among all participants is achieved by sharing the object model, which is comprised of scene object, player object, and environment settings object. The scene object and player object share common properties such as position, orientation, mesh, animation, sound, etc which they inherent from a parent object.

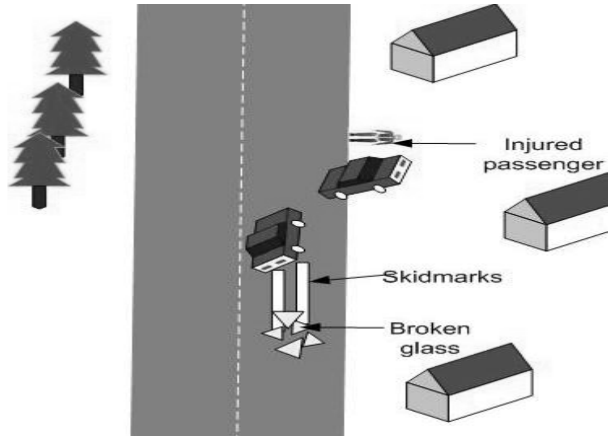

Fig . 4. 2D drawing of the accident scene

Fig . 5. 3D accident scene

Finally, 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 second phase synchronizes the updates to the players and objects properties between all players.

VI. EXPERIMENT

We have developed an accident scenario experiment to measure police trainees' performances while investigating a virtual accident scenario. The scenario involves a crash between two vehicles and one severely injured passenger as shown in Fig. 4 and 5. The roles of the two vehicle drivers are played by two actors who are given the same script for common questions and their answers. If any question is asked by a trainee for which there is no script the actor either replies that he can not remember or invents an answer and that answer is added to the script for the next trainee's experiment sessions. To limit the number of required actors

we gave the role of the operation room to one of the drivers as well. His role was to handle any requests made by the trainee to resolve the incident such as: call an ambulance, request assistance, etc. All the experiments were video taped and analysed afterwards to measure performance.

A trainer at the police academy selected six subjects with similar training backgrounds: four years of police academy and two specialized courses in vehicle investigation. The six subjects were divided into two groups. Group A ran a pen and paper experiment first followed by the computer experiment (each referred to as 'parts' of the pair of experiments). Group B did the opposite. In the pen and paper experiment a 2D drawing of the accident scene is used and the positions of the players are marked by small rectangular pieces of paper which can be moved around. The computer experiment used the prototype developed. The reason for conducting a pen and paper experiment along with the computer experiment is to compare the results to another practical training method.

During each experiment session the subject is expected to go through five investigative stages: receiving incident call, arriving at the accident scene, initial investigation phase, data completion phase, and final investigation phase. In each of the stages the subject is expected to carry out a number of tasks. For example, after arriving at the accident scene the subject has to attend the injured, search for more injured, identify hazards, call the ambulance, call for assistance, etc. The evaluation is done by scoring the completion of successful tasks.

For each subject two performances were measured, one after each experiment. The average performance among all subjects across both experiments (pen and paper, and computer) was at 41.9% (26 out of 62 marks). The average performance result shows that the theoretical and supervised training (previously provided at the academy and police station respectively) on their own are not reaching an adequate performance measure.

The results also show that Group A subjects improved their performances average on the second part by 12.4% compared to 8.1% for the subjects in Group B. In none of the five investigative stages did the pen and paper experiment manage to improve subject performance by more than what the computer experiment achieved.

Another interesting result is the time taken to complete each experiment. We noticed that group B, which did computer experiment first, spent 54.2% less time when doing the following pen and paper experiment. Group A, which did the pen and paper experiment first, spent 36.2% less time when running the following computer experiment. Two initial reasons for this are suggested. Either the computer usage complexity added to the time used or the user started discovering more things in this experiment. The first possibility can be discarded when comparing the average time after the first part for both groups since the pen and paper experiment average time took 3.3 minutes longer than the computer experiment.

Furthermore, during the computer experiments we noticed that all the subjects made good use of the navigation methods and managed to investigate the accident scene. They also made good use of the headset facility in identifying the drivers and communicating with the operation room operator. These two observations back the high rate of presence (76.8%) and co-presence (86.6%) reported by the subjects on a post-experiment questionnaire. Moreover, one of the trainers said that computer experiment training has shown him clearly the trainees' weaknesses which were not obvious to him while conducting the theoretical training during the four year course or the specialized courses at the police academy. During the discussion after the experiment one of the subjects said that this was his first 'severe' accident and that is why he thought he did not perform well. He got the lowest mark of all the participants. His comments indicate that he is relating this experiment and comparing it to his experiences. This confirms the positive responses from all subjects when asked subjectively if this experiment will go into their collection of experiences.

VII. Discussion

This section discusses two issues: the practicality of the prototype architecture and the indications this gives for the implementation of the next phase of the architecture shown in Fig.1.

The prototype had two primary objectives with regards to flexibility: the separation of the domain knowledge and the ability to control it at run-time. The separation has been achieved by the use of a high-level language (Python) to create a layer on top of the simulation environment to act as an interface. This meant that behaviours can 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.

The abilities given to the trainer to create different scenarios by inserting the scene layout, deciding on the number of participants, and specifying the events that occur and allowing him to trigger them at any time have demonstrated the flexibility of the approach since it managed to detach the scenario knowledge from the simulation environment and allow its control at run-time.

The prototype's practicality can be measured on the

following points:

- Ability to accommodate different scenarios
- Amount of simulation engine's functionalities exposed to the trainer at the script level

The accommodation of multiple scenarios is achieved by partially storing some data in the database such as: number of participants, meshes associated with each participant, visibility, scene layout, etc. The other part is done at run-time as the trainer is allowed to set an object's behaviour and trigger events at any time during the session. This illustrates that multiple scenarios can be deployed on the architecture.

The second point is shown in Table 1. It shows the behaviours exposed to the trainer via scripting. What has been exposed is a fraction of the engine's capability resulting in the limited number of things the trainer can do for a scenario. This can be overcome by exposing more functionality.

The developed architecture illustrates how domain-independence can be achieved by following two simple pointers. One is to add a high-level scripting language on top of the simulation environment to allow the addition of scenario logic at any time thus achieving scenario independence. Second is to adopt a mechanism which permits different scenes to be deployed on the architecture without any coding. Furthermore, the suitability of the architecture for training has been demonstrated by the results which show that the environment has permitted trainees to attend and investigate a virtual accident. It also provided the trainer with an environment to evaluate trainees' performances during their investigation. Moreover the high degree of presence and co-presence and the positive comments collected further enhance the training suitability of the architecture.

The prototype has laid the ground work for the next phase of the architecture which investigates two main challenges. The first is embedding a publish/subscribe mechanism in the architecture to promote simulation environment independence (i.e. to service the logic in a generic format that is not limited to one simulation environment or game engine), something which is lacking from the categories described in the related work section.

The second challenge is to build an expert system to take the role of the trainer in generating and controlling scenarios. A knowledge acquisition process has been conducted over a period of two months with experts from the traffic investigation division in Dubai Police and a set of rules have been elicited. The work is now underway to format the rules and build the inference engine which is part of the expert system. The expert system will then be interfaced with the architecture.

VIII. CONCLUSIONS

The work started with two main goals: to create an architecture which is domain-independent and to ease the scenario generation process by automating it. We have detailed the architecture components and how they fit together to achieve these two goals. Furthermore, the prototype developed has shown how the separation of the domain knowledge from the simulation environment can be accomplished. We have shown that such an architecture can handle running multiple scenarios since the sequences of events are placed in the hands of the trainer who can alter the sequences as he wishes. The next stage will replace the trainer by a KB system as illustrated by the architecture proposed in Fig. 1.

ACKNOLWEDGEMENT

This work is sponsored by a grant from Dubai Police. Thanks are due to all personnel from Dubai Police for assisting in the knowledge acquisition phase and taking part in the experiment. Thanks also to Michael Meredith for his support and suggestions that enabled the software to run on the Sheffield VR studio.

REFERENCES

[1] 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, 10th Conference of the European Chapter of the Association of Computational Linguistics, 2003. http://citeseer.nj.nec.com/563862.html

[2] Casanueva, J., 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.

[3] Chi, D., Clarke, J., Webber, B., Badler, N. *Casualty Modeling for Real-Time Medical Training.* PRESENCE: Teleoperators and Virtual Environments Special Issue on The Human Figure in Virtual Environment Systems, Volume 5, Number 4, pp. 359-366, December 1996.

[4] 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.

[5] Dachselt, R. *CONTIGRA Towards a Document-based Approach to 3D Components*. Workshop 'Structured Design of Virtual Environments and 3D-Components' at the ACM Web3D 2001 Symposium.

[6] Davis, W., Moeller, G. *The High Level Architecture: is there a better way*. In Proceeding of the 1999 Winter Simulation Conference.

[7] Drew, R., Morris, D., Dew, P., Leigh, C.*A System Architecture For Supporting Event Based Interaction And Information Access*.http://citeseer.nj.nec.com/370116.html

[8] 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

[9] 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.

[10] 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.

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

[12] Hook, B. *The Secret Life Of Game Scripting*. Feb, 2004 http://bookofhook.com/Article/GameDevelopment/TheSecretLifeofGameScript.html

[13] 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.

[14] 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.

[15] Slater, M. and Steed, A. *A Virtual Presence Counter*. Presence: Teleoperators and Virtual Environments 9(5), 413-434, 2000.

[16] 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.

[17] 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

[18] Tamberend, H. Avocado: *A Distributed Virtual Environment Framework*. Ph.D.thesis, University of Bielefeld, 2003.

[19] 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.

[20] 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.

[21] Wright, I. P., 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.