Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2016, Article ID 8489278, 13 pages
http://dx.doi.org/10.1155/2016/8489278

Research Article

A Dynamic Platform for Developing 3D Facial Avatars in a Networked Virtual Environment

Anis Zarrad

Department of Computer Science and Information Sciences, Prince Sultan University, Riyadh 11586, Saudi Arabia

Correspondence should be addressed to Anis Zarrad; anis.zarrad@gmail.com

Received 1 October 2015; Revised 8 January 2016; Accepted 18 January 2016

Academic Editor: Michela Mortara

Avatar facial expression and animation in 3D collaborative virtual environment (CVE) systems are reconstructed through a complex manipulation of muscles, bones, and wrinkles in 3D space. The need for a fast and easy reconstruction approach has emerged in the recent years due to its application in various domains: 3D disaster management, virtual shopping, and military training. In this work we proposed a new script language based on atomic parametric action to easily produce real-time facial animation. To minimize use of the game engine, we introduced script-based component where the user introduces simple short script fragments to feed the engine with a new animation on the fly. During runtime, when an embedded animation is required, an xml file is created and injected into the game engine without stopping or restarting the engine. The resulting animation method preserves the real-time performance because the modification occurs not through the modification of the 3D code that describes the CVE and its objects but rather through modification of the action scenario that rules when an animation happens or might happen in that specific situation.

1. Introduction

There is a growing interest in online collaborative virtual environment (CVE) applications. CVE applications are as a distributed 3D graphic application where multiple users can interact and collaborate. Each player in the VE is represented by a 3D body called an avatar [1], which allows the players to see, interact with, and hear each other. Many successful applications are already launched in educational, social, gaming, commercial, virtual shopping, and training simulations.

Today, the growth of game engines is accelerating, and the development of VE applications continues to increase. Therefore, there is a need for new approaches that handle VE runtime extensibility requirements without a complex manual initialization and engine restarting. Such a requirement is vital for critical virtual environmental applications like military training, emergency preparedness scenarios, and E-shopping. The integration of the avatars' face animation as an on the fly feature will complicate the task and require an important amount of work for qualified artists with a strong knowledge in facial anatomy.

The ability to change an application without having to stop it is an important nonfunctional requirement for 3D CVEs especially if they are to be used as disaster management systems, which should be available around the clock. In CVE, changes in the CVE requirements are driven by changes in the game engine and are time-consuming. Broadly, 3D objects and avatars' faces may change in various ways to give the users a visual display about the actions that are being applied to the objects or embed animation modeling into an avatar according to the constantly changing situation during a disaster. Traditionally, any modifications to the virtual environment system require collaborative effort from graphic designers and 3D programmers to develop the required new scenario and then stop and restart the engine to reflect the new modification in the VE. Due to its complexity, we focus here on avatar facial animations. Several research studies on 3D disaster management have been used to model human behavior and offer true-to-life VE. However, there is still a lack of studies that can easily present avatar animation in emergent situations [2–5] and in real time without stopping the engine. Therefore, 3D modifications can be very time-consuming and costly.

The challenge that the 3D game engine had to deal with is the change of the avatars behavior during the game scenario. We propose a framework that implements a modular architecture based on atomic simulation concepts to manage both the virtual environment content and avatars behavior in dynamic environments and guarantee continuity during runtime. The proposed game engine is designed to separate the language used to implement the game from the game scenario and offer flexibility and simplicity to the user when modification is required. We integrate a scripting story interpreter component in the game engine to control and manage avatar face animation changes. We found that using an eXtensible Virtual Environment Markup Language (XVEML) as a general-purpose of event-based state machine language can make the development easier and faster during the runtime and discharge designers and professionals' programmers.

The XVEML provides hierarchical structure content for simulation description, objects behaviors, and avatar facial animation. We choose MPEG-4 Facial Animation Parameters (FAP) for avatar facial animation [6, 7]. We used VRML to define the avatars' face because VRML provides a standardized functionality and is also aligned with the MPEG-4 standard [2]. We adopt two script levels called *Class* and *Instance* for fast creation of large-scale VE applications. In the *Class* level we incorporate the atomic simulation [8] concept to model all possible simulation scenarios. In the *Instance* level, we incorporate the atomic behavior and action concepts to manage entities behavior and avatars' facial animation. Two types of state machines can be distinguished in our system—those modeling avatars and entities (mainly avatars' facial expression) and those modeling the simulation scenario. Consequently, two script files called Scenario Simulation XML (SCXML) and Instance XML (IXML) are generated automatically and independently based on each state machine modeling. The use of two separate scripting levels offers more flexibility to our solution and allows simulation flow, object specifications, and avatar behavior to be managed independently.

In BIM (building information modeling) [5] a significant number of developer and graphic designers are needed to represent human behavior in a real-time game environment for fire evacuation to conduct effective information interaction between the building and the avatars. Many previous studies have completely discarded avatars' behavior and animation [3, 9] concepts. In 3D disaster management applications, the avatar behavior is necessary to offer full realism for effective interpersonal communication on a level of richness interchangeable with face-to-face interactions.

Game engine systems have recently been in the popular press. Before the appearance of game engine technologies [10–12], most systems were developed as virtual reality systems to handle specific task such as NPSNET [13], DIVE [1], and SPLINE [14]. Thus, any modification requires a hard change in the programming environment and architecture. As game engine technology matures and becomes more flexible to 3D environments, programming skills remain a concern and can hamper the development of complex environments. Existing programming approaches such as VRML [4], OpenGL [15], X3D [16], and MPEG-4 [2] can build CVE applications. However, most of them cannot provide native support for such systems. Consequently, extensibility action requires complete loading of the CVE application into memory before the modification occurs.

The remainder of the paper is organized into five main sections. In Section 2, we provide some related works. An overall system architecture is presented in Section 3. Section 4 describes the novelty in our proposed approach to build and/or extend a VE application as avatar facial expression during runtime. In Section 5 we propose a firefighter case study. An overview of the proposed XVEML language is given in Section 6. An extensity scenario example is provided in Section 7. Section 8 describes the modular architecture used in our system and prototype implementation. We conclude with a discussion and future direction of system development.

2. Related Works

CVEs are increasingly attractive especially in education, entertainment, simulation, and many others. Today, the research tendency leans towards easy and rapid runtime without having intensive programming skills.

Most game engines use the Unity engine [10], the Unreal engine [17, 18], Gamebryo engine [12], CryEngine [19], and Software's Source engine [11]. However, these are limited to specific tasks, and their features are coupled with proposed game characteristics. Thus, any extension or change that adopts a new feature in the application requires a game engine restart. Such systems are promising platforms as long as they serve a specific application without extensions. In addition, only professional programmers can modify the virtual environments within games because they are complex. Choosing an adequate game engine depends on the goal, platform, and speed with which changes are needed.

Wang et al. [5] developed a BIM game based on virtual reality to provide real-time fire evacuation guidance. However, modification is static and limited, which means that it cannot dynamically change the 3D content according to a constantly changing situation during a fire emergency. In addition, a significant number of developers are needed to design and implement this system. In [20], the authors introduced a fire-training simulator to allow trainees to experience a realistic fire scenario and assess different rescue plans in a graphic environment. Representations of human behavior are completely ignored in the implemented case studies. This influences the validity and the true-to-life concept of the VE. Cao et al. [21] presented complex real-time facial recognition using 3D shape regression. The animation algorithm uses a set of training data generated from 2D facial images. System accuracy is improved when the captured image and training data are increased. This requires considerable data to process and may lead to failure in real time.

Before game engine, virtual environments were developed using dedicated systems to implement a specific scenario. Some of the most well-known former systems are DIVE [1], MASSIVE [22], NPSNET [13], SPLINE [14], and

VLNET [23]. They focus on particular applications to reduce the overall implementation complexity. The problem stems from the fact that systems are strongly coupled in terms of implementation. Consequently, any modifications in the application require modifications in the supporting architecture because the complexity is due to a combination of the internal architecture and specific application functionalities.

Zarraonandia et al. [24] described a 3D virtual environment to improve the learning of airport emergency protocols. Each user plays a different role in a particular emergency situation. The idea is based on replication and does not reflect a real context. In [25], the authors proposed a solution to help specialists and decision makers better understand, analyze, and predict natural disasters to reduce damage and save lives. One important factor that is not taken into consideration is the dynamic behavior of the disaster. System interruption is required to adopt new scenarios.

To support extensibility, several approaches have been implemented. Magerko and Laird [26] used a microkernel-based architecture to separate the system elements from the kernel to add, remove, and modify during runtime. Unfortunately, this highly accredited approach incurs a great deal of complexity particularly in terms of facilitating communication between components written in different languages. Also, extensibility requires an intensive knowledge of programming language. Oliveira et al. [9] developed a Java Adaptive Dynamic Environment (JADE) based on the Java architecture. It consists of a lightweight cross platform kernel that permits system evolution during runtime. The adoption of JADE does not provide an efficient solution to problems from extending CVE applications.

In [3, 27], the authors developed a Virtual Environment Markup Language (VEML) based on the nonlinear story concept defined by Szilas [28] to build and/or extend CVE applications. This model allows story progress during the simulation, which is implemented independently of the 3D environment programming. In [3], a repetitive atomic simulation was modeled in response to similar events. For example, in virtual shopping, the Real Madrid FC store may run many sales around the year (every game). To control this situation, VE manager will send a VEML script file to all participating clients whenever a sale event is set. Thus, by transferring many descriptions, the files regularly impact the network bandwidth. Also, any change in the script file must be done manually. This may lead to unplanned atomic simulation and the loss of realistic appearances of the application.

Many other systems have adopted script language technology to design avatar facial and body gesture animations. The Virtual Human Markup Language (VHML) [28] is an independent XML-based language used in MPEG-4 applications which encapsulates a markup language dedicated to body animation (BAML). Perlin and Goldberg [29] describe an IMPROV system using a high-level script language for the creation of real-time behavior with nonrepetitive gestures. Arafa et al. [30] describe an avatar markup language AML based on XML to encapsulate text to speech as well as facial and body animations in a unified manner with appropriate synchronization. The main concern in the proposed work is the complexity implementation that requires realistic avatar facial animation. Furthermore, their supporting architectures are typically designed to handle only avatar behaviors. As a result, it is difficult to apply any relevant code from a particular system and adapt it to our target system. In addition, implementation and change in the code are done manually and require a change in the supporting structure.

3. System Architecture

This work is a part of a funding project from a Research and Translation Center (RTC) in Riyadh, Saudi Arabia. The goal of the project is the design and implementation of a complete 3D collaborative virtual environment application for disaster management. It is important to have a believable 3D virtual environment to prepare a rescue scenario with an acceptable response time whenever there is a need. Having such system with a 3D representation of current situation as input is very helpful for police, military, and medical staff in order to let them react properly with an appropriate effort management when they arrive to the accident site.

The overall scenario can be described as follows: once deployed sensors in the monitored area detect a fire disaster or crowded zone, a protocol for gathering data like location and severity is set. The data is sent to the central location through a communication mechanism between sensor node and central station. Many robots are sent to the accident site to investigate site and send more details to the central station. Based on the received data, a new 3D representation is created to closely reflect the reality.

Depending on the situation, a rescue plan is established using the IA mechanism to give workers (firefighters, police, etc.) the quickest access with improved response. Rescue implementation will vary according to the work situations, the equipment used, and the victim's condition. An implemented rescue plan is governed by the on-site rescue plan. In this work we focus only on the avatars facial behavior during the runtime to reflect the real situation described in the sensors data. The overall system architecture is presented in Figure 1. Three layers are identified: (1) the WSN layer, (2) the 3D virtual environment server layer, and (3) the client side 3D player.

Initial 3D representation of the VE is preloaded into the server for quick access and modeling. We focus only on the game engine components implemented in the 3D server layer to manage facial animation changes in the VE when there is a need for requirements adjustment. Wireless sensors protocols and the artificial intelligence (AI) mechanism to establish the rescue plan are out of the paper scope. Engine architecture is detailed in Section 8.

The implementation of the main game engine components in the server side offers the user the freedom to use any device (desktop, mobile devices, etc.) when operating the system.

4. Proposed Approach

In addition to advancing our understanding of the factors that contribute to the 3D disaster management application

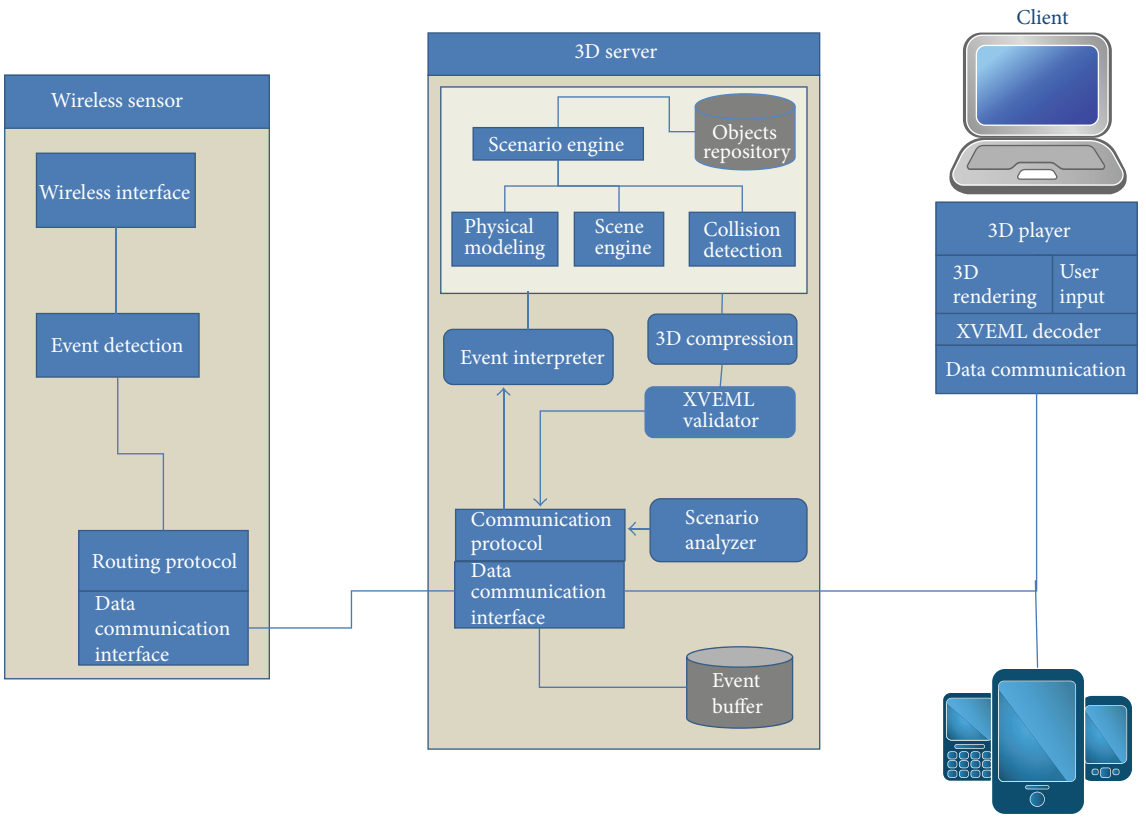

Figure 1: The overall system architecture.

platforms, the results are useful for other applications such as military training and virtual shopping via an automatic scenario for face expression adaptation. Disaster management is a dynamic and delicate environment where many events can occur suddenly in real life. They must be reflected in the 3D environment to enable better performance in live exercises and real incidents.

In our previous work [3], we described a script language through the concept of atomic simulations. The resulting file language is generated manually and emphasizes the simulation scenario and completely ignores the avatars' facial animations and objects' behaviors. In this way, the file generation process becomes tedious, and the scenario becomes complex. The number of avatars participating in the VE is very large.

In conclusion, a lack of facile tools limits freelance developers. Thus, our goal was to overcome the complexity of established modeling tools by combining them with the concept of atomic behaviors. Our methodology models the avatars and objects separately from the simulation execution scenario to consider the importance of the avatars' behavior.

4.1. State Machines Modeling. We propose to design the entire virtual environment simulation scenario and its entities (avatars facial animations, 3D geometric objects, and their behaviors) through the state machine theory. We use a branching graph story [31] incorporated with atomic model concepts. Through atomic modeling, we mean both atomic simulation and atomic action including avatar facial animation and object behavior. We use an atomic simulation similar to [8]. Atomic avatar facial animation is a primitive state that cannot be divided further. It is needed for building complex facial expressions, for example, talking with the head moving. Each atomic model runs as a distinct process to facilitate the modification. As a result, the user and designer can make any modifications in the simulation scenario without affecting the facial animations and vice versa. The branching graph takes the form of directed graph containing nodes and arcs between nodes. In this approach, we define chapters as a set of atomic simulations, and a node denotes a chapter. Some nodes exist outside the simulation graph to denote the planned set of avatars facial animation in a specific chapter and can be activated by precondition rule-based or without any incoming transition.

We denote $E_{i,\text{avj}}$ as a facial animation expression for a designed avatar with ID avj:

$$i \in \{\text{“Happy”}, \text{“Frightened”}, \text{“Stressed”}, \text{“Neutral”}\}. \quad (1)$$

Each chapter can be designed through a state machine which is defined as a set:

$$A = \{S, E, T, S_0, F\}, \quad (2)$$

where S is a set of finite states, E is an alphabet, T is a transition function $T : S \times E \rightarrow S$, S_0 is initial states, and F is final states.

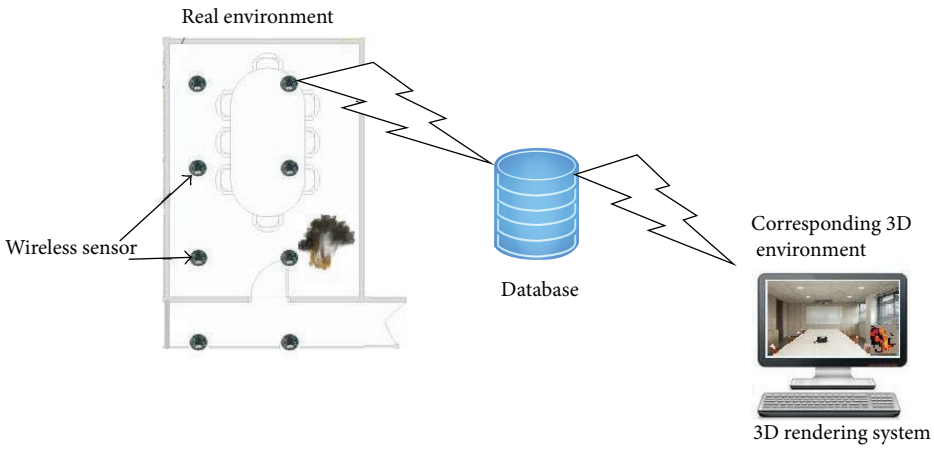

Figure 2: Fire scenario in meeting room.

4.2. Script Languages. The proposed script language is retrieved directly from the state machines as output. Two separate script files are generated to make the VE extensibility procedure more rapid and easy in large-scale VE systems as follows:

(i) The first script language file is called Scenario Simulation XML (SCXML) and it is used to model the scenario simulation (scene description and objects) and predefined avatar facial animation; it is class modeling. This file is less susceptible to changes. However, when any modification has to be reported to the scenario (delete a chapter, add a new chapter, etc.) or to the facial animation (define a new avatar expression and modify an expression), it can be done easily through our state machines. A new script language will be generated and will be injected to the environment. Moreover, the modified script is uploaded into the VE application during the runtime without the need to stop the application.

(ii) The second script language file is called Instance XML (IXML). Here, all the object properties are set to defined values: 3D geometry objects specifications, avatar representation (ID, type, etc.), and the assigned facial animation for a specific avatar. For example, avatar 2 will be happy at time point 00:00 in chapter X. This script language is a class instance for a given scenario.

Clearly the use of two separate modeling ideas is very promising because the VE changes are more frequent in objects and avatar behaviors rather than the simulation scenario. Thus, there is no need to modify our entire VE when the 3D object behavior or avatar facial animation needs to be changed.

5. Case Study: Firefighters Emergency Preparedness Scenario

To better understand the impetus of our proposed approach, we illustrate a true-to-life firefighter emergency preparedness scenario in a meeting room that is used for training purpose. The overall scenario is presented in Figure 2.

Wireless sensors are deployed in the monitoring area (meeting room, real environment). An initial 3D environment covering the monitored area is loaded in the server priorly. When sensors detect a fire situation in this real environment, many events are collected and are sent back to a central database.

A rendering protocol is used to modify the initial 3D environment in order to visualize the detailed live representation in the meeting room and everything that is going on within the target environment including the new events such as fire and smoke.

To deal with the fire disaster, many human resources are involved including firefighters, fire truck drivers, police officers, and ambulance services. The Emergency Service (ES) receives an emergency call from a witness reporting a fire disaster somewhere in the city. ES informs the closest fire station (FS) having sufficient efforts. In addition we may inform ambulance service and police in case there are injuries. Players take the role of witnesses, firefighters, police officers, and medical staff to deal with the fire. Based on the fire density we may need to call two fire stations. Table 1 shows the story structure.

When the game begins, the fire location is not known. FS1 has higher effort than FS2. For simplicity, effort is measured based on the number of firefighters. The game initializations are as follows: the number of fire stations is 2 (FS1 and FS2), the number of firefighters in FS1 is 5, the number of firefighters in FS2 is 3, the number of medical staff is 2, and the number of policemen is 2.

After defining the simulation scenario and its chapters as described in Table 1, we modeled the complete scenario using a state machine. Each chapter is modeled separately using a state machine concept. Every state has a label, which will help us to determine, at any moment, in which state of which chapter our scenario is. Some states are blocking because they need a specific external event. For example, the state "*PrepareEquipment*" in chapter 3 is waiting for the event "3000" (an event can be modeled as a simple OS signal),

Table 1: Fire scenario.

Chapters	Chapter details
1	*Fire scene*
1.1	Observe scene by witness, witness face expression: *frightened*
1.1.2	Get location info
1.1.3	Estimate the fire density
1.2	Call emergency center, witness face expression: *stress*
2	*Emergency center*
2.1	Ask the closet fire station (FS)
2.1.1	FS1 identified
2.1.1.1	Firefighters availability indicates weak efforts
2.1.1.2	Firefighters availability indicates good efforts
2.1.1.3	Firefighters availability indicates excellent efforts
2.1.2	FS2 identified
2.1.2.1	Firefighters availability indicates weak efforts
2.1.2.2	Firefighters availability indicates good efforts
2.1.2.3	Firefighters availability indicates excellent efforts
2.2	Call the police center
2.3	Call the ambulance service
3	*Fire station (one selected)*
3.1	Prepare equipment
3.1.1	Firefighter inside the trucks, firefighters face expression*: stressed (all)*
3.2	Move to location
3.3	Manage the fire
3.3.1	Set equipment
3.3.2	Deal with the fire
3.3.2.1	FS1.AV1, FS1.AV2 evacuate people, face expression: *happy*
3.3.2.2	FS1.AV3, FS1.AV4 extinct fire, face expression: *frightened*
3.3.2.3	FS1.AV5 control equipment, face expression: *stressed*
4	*Fire station (two selected)*
4.1	Prepare equipment
4.1.1	Firefighter inside the trucks
4.2	Move to location
4.3	Manage the fire
4.3.1	Set equipment
4.3.2	Deal with the fire
4.3.2.1	FS2.AV1 evacuate people, face expression: *happy*
4.3.2.2	FS2.AV2 extinct fire, face expression: *frightened*
4.3.2.3	FS2.AV3 control equipment, face expression: *stressed*
5	*Police service arrives*
5.1	Clean the area
5.1.1	Control the local traffic
5.1.2	Secure the fire site
5.2	Investigate the area
5.2.1	Ask witness, witness face expression: *stressed*
6	*Ambulance service arrives*
6.1	Manage evacuated people, staff face expression: *stressed (all)*
6.1.1	Injuries identified, staff face expression: *compassionate (all)*
6.1.1.1	Provide first-aid on site
6.1.1.2	Transport to hospital
6.1.2	No injuries identified, staff face expression: *happy (all)*
7	*Fire extinct*
7.1	By FS1, firefighters face expression: *happy (all)*
7.2	By FS2, firefighters face expression: *happy (all)*
7.3	By FS1 and FS2, firefighters face expression: *happy (all)*

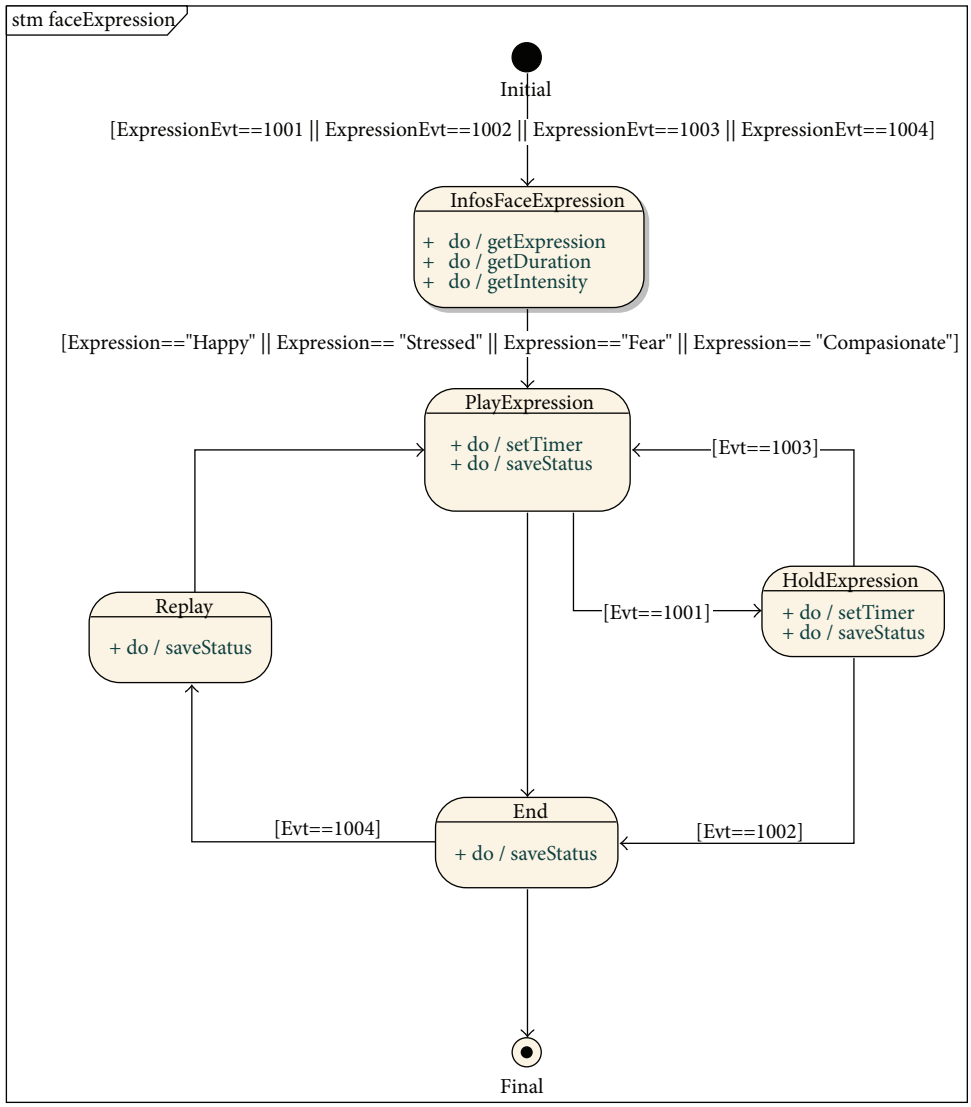

Figure 3: Avatars face animation modeling.

which will be emitted by a state in another chapter as shown in Figure 3.

In this scenario, we focus on the avatars facial animation expressions. The avatar dynamically changes his behavior to reflect the surrounding situation in the virtual environment depending on the current situation and what is being executed.

In Figure 3, we present facial animation modeling. Any facial animation is defined with specific attributes such as a file name describing the animation type and two parameters, duration and start time. Duration is used to define the duration of the animation (e.g., happy for 25 seconds). Start time parameters are used to define when the animation should start.

It is important to notice that any animations can be planned for a replay when there is a need without the involvement of a VE designer in each replay. Such modeling offers a better solution especially for an application that requires regular modifications. For example, doctor should be happy when meeting patient in E-health application or during end of year sale in E-shopping application where many red sale signs should be integrated into the 3D E-shopping.

6. Markup Language VEXML

In this section we explore the syntax and elements of the proposed XVEML language. Algorithms 1 and 2 show screenshots of the language. It is composed principally of SCXML and IXML. The root tag <XVEML> marks the beginning and end of the script.

6.1. Syntax and Simulation Scenario. Algorithm 1 gives an overview of our SCXML script language file. The original file has been relieved of some irrelevant tags. The resulting script

```
<?xml version="1.0" encoding="UTF-8"?>
<XVEML>
<connector xmi:idref="EAID_11A4687E_B5B3_451e_95F1_0CAF77327D4D">
<source xmi:idref="EAID_85BFDD36_7149_43b8_8F62_7A0EBE5D338F">
<model ea_localid="54" type="State" name="MoveToLocation"/>
</source>
 <target xmi:idref="EAID_9F1539F4_2DF9_40db_8FE5_0E6F7B8EB2AB">
 <model ea_localid="55" type="State" name="ExtinctFire"/>
 </target>
<labels mt=" [Expression==1]"/>
 </connector>
<connector xmi:idref="EAID_42C7882B_A52A_4a2e_A03B_63DC132DFBAD">
<source xmi:idref="EAID_E5E29AA8_0921_444f_9666_EC47340F9417">
<model ea_localid="53" type="State" name="PrepareEquipment"/>
</source>
 <target xmi:idref="EAID_85BFDD36_7149_43b8_8F62_7A0EBE5D338F">
 <model ea_localid="54" type="State" name="MoveToLocation"/>
 </target>
<labels/>
</connector>
 <connector xmi:idref="EAID_82DBB96D_4094_4428_AD2C_0EE4C8848FC5">
 <source xmi:idref="EAID_C0E3E4C7_ECED_48d3_9478_0E4720678FC9">
 <model ea_localid="52" type="StateNode" name="Initial"/>
</source>
 <target xmi:idref="EAID_E5E29AA8_0921_444f_9666_EC47340F9417">
 <model ea_localid="53" type="State" name="PrepareEquipment"/>
</target>
<labels mt=" [Evt==3000 || Evt==4000]"/>
</connector>
</XVEML>
```

Algorithm 1: SCXML script file.

language is organized as set of "connectors" which mean two states: "Source" and "Target." Each state is defined by its name and its ID. An extra tag "Label" is used when there is a transition from "Source" to "Target" that meets some prerequisite conditions.

The root tag <AFSL> marks the beginning and the end of Avatar Facial Script Language. It accepts four attributes: avatar face ID (*unique reference ID for the 3D face to be animated*), file path (*the path for animated file used in the animation*), start time (*the time where the animation must start*), and description (*describes the animation*).

Expressions tag has two subelements <TTS> or text to speech and <Expression>. There may be many Expressions and only one <TTS>; text to speech cannot be overlapped. The Tag TTS accepts three attributes: start time (starting time of the animation), fab file (file used to generate the Facial Animation Parameters FAP output), and Audio File (file used to generate the audio output of the text to be spoken). Expression may or may not overlap.

Each <Expression> has a start time, repeat number (to specify the number of repeats in need, otherwise default value 0), pause time (when a repeat is needed, we specify the pause time between consecutive expressions), and animation description. We choose this scheme to define expressions using a set of distinct expression to allow the designer to add as many expressions as desired from the FAP database. However, only one TTS is allowed in Expressions to control the overlapping.

7. Extensibility Mechanism at Runtime

Changing and/or extending the existing VE application during the runtime must be done easily and smoothly. Most systems require extensive programming activities and collaboration with different professional to manipulate the extended action. The dual modeling of our VE application allows us to rapidly and easily prevent any changes in the VE simulation during modeling. For example, as shown in Figures 2 and 3, if we need to modify the simulation scenario by dropping chapter B from the global scenario, we have to modify the corresponding state machine by removing chapter B and then automatically generate the new SCXML file. Consequently, when the simulation scenario changes, it must be followed by instances modification. Thus, there is a need to generate a new IXML file as well. On the contrary, any modifications in instance models do not require a modification in the class model. Such an approach makes code that would otherwise be mitigated and challenging to modify simple, easy, and accessible to novices. In addition, the challenges of developing SCXML and IXML script codes

```
<IXML>
<xmi:Extension extender="...">
<declaration>
<Object name= "AV1" file = "Firefighter.mp4" manager= "Keyboard"/>
<Expression name = "Happy" File = "Happy.mp4"/>
<TTS start="t2" fab_file ="ABS" wav_file="SS" Text="Text to speak">
</declaration>>
<elements>
<element xmi:idref="EAID_...2C38" xmi:type="uml:State" name="ControlEquipment" scope="public">
<EAModel.scenario>
<EAScenario name="Expression ( FS1.AV5, &quot;Stressed&quot;, StartTime = 12, Duration=30,
Intensity= &quot;Medium&quot;) " type="Alternate"../>
</EAModel.scenario>
</element>
<element xmi:idref="EAID_...7E34" xmi:type="uml:State" name="EvacuatePeople" scope="public">
 <EAModel.scenario>
<EAScenario name="Expression ( FS1.AV1, &quot;Happy&quot;, StartTime = 12, Repeat = n,
PauseTime= p, Duration=40, Intensity= &quot;Low&quot;) />
<EAScenario name="Expression ( FS1.AV2, &quot;Happy&quot;, StartTime = 32, Duration=40,
Intensity= &quot;Low&quot;) " type="Alternate" />
</EAModel.scenario>
</element>
<element xmi:idref="EAID_...B2AB" xmi:type="uml:State" name="ExtinctFire" scope="public">
......
</element>
<element xmi:idref="EAID_...338F" xmi:type="uml:State" name="MoveToLocation" scope="public">
 <EAModel.scenario>
<EAScenario name="MoveTo(FS1.AV1,&quot;&quot;)" type="Simple" weight="1,00"
subject="EAID_...338F" xmi:id="EAID_...C3AD"/>
....
</EAModel.scenario>
</element>
<element xmi:idref="EAID_...9417" xmi:type="uml:State" name="PrepareEquipment" scope="public">
<EAModel.scenario>
<EAScenario name="Expression ( FS1.AV1, &quot;Stressed&quot;, StartTime = 12, Duration=30,
Intensity= &quot;Low&quot;) " type="Alternate"../>
.....
</EAModel.scenario>
</element>
<element xmi:idref="EAID_...056A" xmi:type="uml:StateNode" name="Final" scope="public">
<extendedProperties tagged="0" package_name="Chapter3_4"/>
</element>
<element xmi:idref="EAID_...8FC9" xmi:type="uml:StateNode" name="Initial" scope="public">
<extendedProperties tagged="0" package_name="Chapter3_4"/>
</element>
</elements>
 </xmi:XMI>
</IXML>
```

Algorithm 2: IXML script file.

for execution in large-scale VE application can completely avoid limitations in traditional development processes. The resulting files are sent to all users participating in the VE during runtime without interrupting the system.

In addition, there is a need to manipulate existing avatars in the dropped chapter to guarantee in a natural and realistic way the developed application. Two approaches can be proposed: managing the singleton manually by directly changing the avatar attributes in the objects' repository database or creating specific state machines (migration state machine) that will remove the avatars from the ghost chapter B to any other chapter.

8. Game Engine Architecture and Prototype Implementation

The proposed architecture is specifically designed to bridge the gap between extending the 3D VE during runtime and

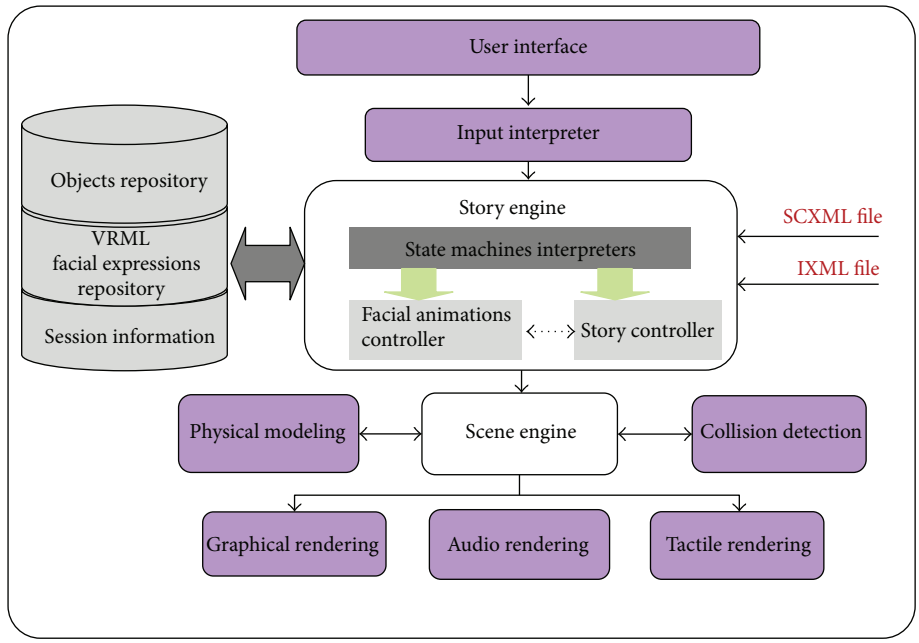

Figure 4: Proposed game engine architecture.

Figure 5: Face animation control.

the system functionality interruption. We use modular architecture to facilitate the integration and the management of new components and offer qualities to the developed system. In client side architecture, the model as well as animations engines are web-based, and animation results can be viewed locally in the end user browser. Figure 4 shows the detailed proposed architecture.

The low level architecture deals with the extensibility at runtime and is composed of the following:

(a) Message controller: it acts as an entry point to receive a message from the server side and identifies the set of message that can be accepted.

The high-level architecture is composed of the following:

(a) User interface: it is composed of GUI and VRML viewer. *GUI* stands for the graphical user interface of the system to transmit the user's commands such as the following: modify the scene content or add avatar face animation. *VRML viewer* visualizes the virtual environment content and the VRML avatars' face animations.

(b) Story engine: it is the main component in this architecture. It models the simulation scenario of the proposed system and avatar facial behavior through the state machine concept. This component is composed principally of the following.

(i) *State Machines Interpreter*. In a classic case, this module will get in its input only once from a SCXML file (the scenario story). It models the states and transitions and manages all possible transitions through the defined states. There are several examples of the IXML file, which represent one instance for the given scenario. Also, this module is responsible for the validation of the files using a parser.

(ii) *Story Controller*. This is a component responsible for converting the SCXML into commands for the story engine.

(iii) *Face Animation Controller*. It is a component responsible for managing the IXNML file and performing core functionalities of the execution of facial animations.

(c) Conflict detection: it will address the conflict issue in facial animation such as when two animations cannot be combined (happy expression and angry expression).

(d) VRML faces expressions and animation database: it contains a description of a face. Each facial object

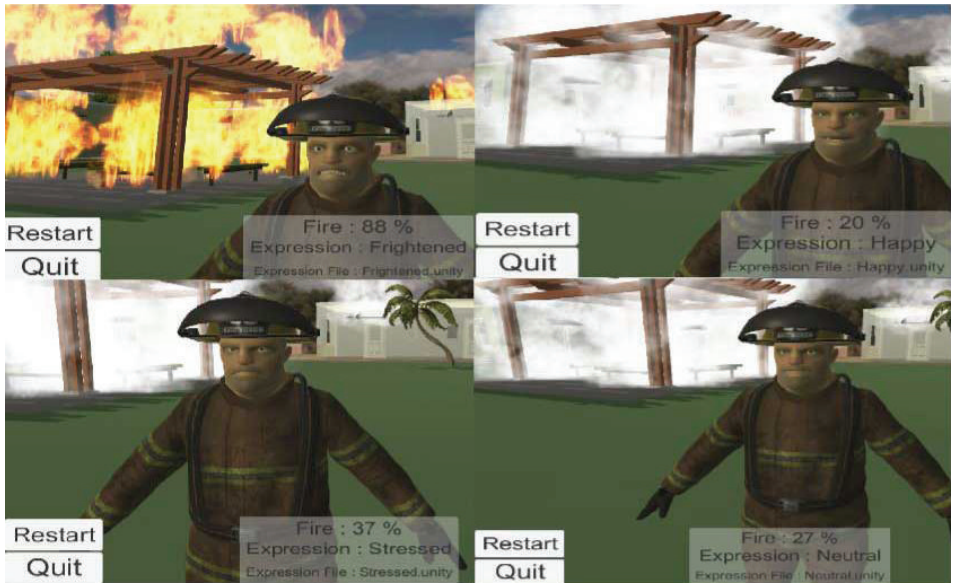

Figure 6: Firefighter face animation.

is implemented as a VRML transform node, which describes shapes and other properties. This also has all new facial animations created during the simulation.

Avatars participating in this scenario are modeled with four animations: frightened, stressed, neutral, and happy depending on the situation context.

The XVEML file is first written by serializing the expression data. We then write into the file using the standard C# IO functions:

```
var serializer = new XmlSerializer(typeof
(xExpressionList));

var stream = new FileStream
("ExpressionList.xml", FileMode.Create);

serializer.Serialize(stream,
expressionList);
```

Figure 5 shows the firefighter face modeling and the animation control features. We used sixteen features marked as black dots on the face. Texturing was done using Photoshop.

For easy facial animation we adapt the Rigging Process [32]. We have built a "Custom Controller" for every feature to get a real avatar emotion in the Face Animation Controller module. Figure 6 shows four animations (frightened, happy, neutral, and stressed) depending on the fire density in the actual scene. The corresponding XML file structure is as in Algorithm 3.

When fire density becomes low, the firefighter changes his expression to happy. The State Machine Interpreters module communicates events to the Face Animation Controller to be reflected on the firefighter face. Appropriate face expression file is loaded for rendering.

```
<ExpressionList>
  <Expression>
    <Firefighter_ID>1</Firefighter_ID>
    <ExpressionName>Frightened</ExpressionName>
    <ExpressionFile>Frightened</ExpressionFile>
              <Parameters>
                  .......
              </Parameters>
</Expression>
<Expression>
    <Firefighter_ID>2</Firefighter_ID>
    <ExpressionName>Stressed</ExpressionName>
    ..............
    </Expression>
<Expression>
    <Firefighter_ID>2</Firefighter_ID>
    <ExpressionName>Happy</ExpressionName>
      <Parameters>
              <Replay value="No" >
              <Start value="10">
              .....
      </Parameters>
</Expression>
</ExpressionList>
```

Algorithm 3

9. Conclusions

In this work, we present a new architecture to develop and extend VE applications with facial modeling without game engine interruption during runtime. This approach offers benefits to human life especially when used for critical actions that require an immediate update like E-health and military training applications.

We describe our game scenario as a nonlinear story using state machine concepts. Avatars and object behaviors are modeled separately to offer more flexibility and robustness when updates are required. The script file describing facial animation is retrieved directly from the state machine module and injected on the fly in the game engine. Generated scripts can be reused and customized to fit a wide range of other virtual environment applications such as military training, E-learning, E-shopping, and E-health. However, the system still has some limitations for further improvement in the validation process about facial features in complex scene.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The author would like to acknowledge the support of Research and Translation Center (RTC) in Prince Sultan University, Saudi Arabia. This work was supported by Grant no. GP-CCIS-2013-11-10 from Research and Translation Center.

References

[1] O. Hagsand, "Interactive MUVEs in the DIVE system," *IEEE Computer*, vol. 3, no. 1, pp. 30–39, 1996.

[2] Information Technology—Coding of audio-visual objects—Part 1: Systems, ISO/IEC JTC 1/SC 29/WG 11-14496-1, 2000.

[3] A. Boukerche, D. Duarte, R. Araujo, L. Andrade, and A. Zarrad, "A novel solution for the development of collaborative virtual environment simulations in large scale," in *Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real Times (DS-RT '05)*, pp. 86–97, Montreal, Canada, October 2005.

[4] VRML Homepage, 2012, http://www.w3.org/MarkUp/VRML/.

[5] B. Wang, H. Li, Y. Rezgui, A. Bradley, and H. N. Ong, "BIM based virtual environment for fire emergency evacuation," *The Scientific World Journal*, vol. 2014, Article ID 589016, 22 pages, 2014.

[6] J. Noh and U. Neumann, "A survey of facial modeling and animation techniques," Tech. Rep. 99-705, USC, 1998.

[7] I. S. Pandzic and R. Forchheimer, *MPEG-4 Facial Animation: The Standard, Implementation and Applications*, John Wiley & Sons, New York, NY, USA, 2002.

[8] A. Boukerche, D. D. Duarte, and R. B. de Araujo, "A language for building and extending 3D virtual web-based environments," in *Proceedings of the 2th Latin American Web Congress and the 10th Brazilian Symposium on Multimedia and the Web (WebMedia-LA-Web '04)*, pp. 114–116, IEEE, Ribeirão Preto, Brazil, October 2004.

[9] M. Oliveira, J. Crowcroft, and M. Slater, "Component framework infrastructure for virtual environments," in *Proceedings of the 3rd International Conference on Collaborative Virtual Environments (CVE '00)*, pp. 139–146, ACM, San Francisco, Calif, USA, September 2000.

[10] Unity, http://www.unity3d.com.

[11] Valve Software, Source engine, 2015, http://source.valvesoftware.com/.

[12] Emergent Game Technologies, Gamebryo, 2015, http://www.gamebryo.com/gamebryo.php.

[13] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz, "NPSNET: a network software architecture for large scale virtual environments," *Presence: Teleoperators and Virtual Environments*, vol. 3, no. 4, pp. 265–287, 1994.

[14] SPLINE Homepage, http://www.merl.com/projects/spline/.

[15] OpenGL Homepage, 2015, http://www.opengl.org.

[16] Web3D, 2012, http://www.web3d.org.

[17] T. Sweeney, *Unreal Script Language Reference*, 1998, http://unreal.epicgames.com/UnrealScript.htm.

[18] Epic Games. Epic games' unreal development kit eclipses 50,000 users in one week, 2015, http://www.udk.com/udk50k.

[19] CryEngine, 2015, http://www.crytek.com/cryengine.

[20] A. Ren, C. Chen, and Y. Luo, "Simulation of emergency evacuation in virtual reality," *Tsinghua Science and Technology*, vol. 13, no. 5, pp. 674–680, 2008.

[21] C. Cao, Y. Weng, S. Lin, and K. Zhou, "3D shape regression for real-time facial animation," *ACM Transactions on Graphics*, vol. 32, no. 4, article 41, 2013.

[22] S. Benford and L. Fahlén, "A spatial model of interaction in large virtual environments," in *Proceedings of the Third European Conference on Computer-Supported Cooperative Work 13–17 September 1993, Milan, Italy ECSCW '93*, pp. 109–124, Springer, Berlin, Germany, 1993.

[23] S. Pandžić, K. Tolga, L. Elwin, N. Thalmann, and D. Thalmann, "A flexible architecture for virtual humans in networked collaborative virtual environments," in *Proceedings of the Eurographics*, pp. 177–188, Budapest, Hungary, 1997.

[24] T. Zarraonandia, M. R. R. Vargas, P. Díaz, and I. Aedo, "A virtual environment for learning aiport emergency management protocols," in *Human-Computer Interaction. Ambient, Ubiquitous and Intelligent Interaction*, vol. 5612 of *Lecture Notes in Computer Science*, pp. 228–235, Springer, Berlin, Germany, 2009.

[25] L. Hashemi Beni, M. A. Mostafavi, and J. Pouliot, "3D dynamic simulation within GIS in support of disaster management," in *Geomatics Solutions for Disaster Management*, Lecture Notes in Geoinformation and Cartography, pp. 165–184, Springer, Berlin, Germany, 2007.

[26] B. Magerko and J. E. Laird, "Building an interactive drama architecture," in *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, pp. 24–26, Darmstadt, Gemany, 2003.

[27] A. Zarrad and A. Bensefia, "A novel approach to develop large-scale virtual environment applications using script-language," in *Proceedings of the 9th International Conference on Innovations in Information Technology (IIT '13)*, pp. 169–174, Abu Dhabi, United Arab Emirates, March 2013.

[28] N. Szilas, "IDtension: a narrative engine for interactive drama," in *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE '03)*, Göbel, Ed., Fraunhofer IRB, Darmstadt, Germany, March 2003.

[29] K. Perlin and A. Goldberg, "Improv: a system for scripting interactive actors in virtual worlds," in *Proceedings of the Computer Graphics Conference (SIGGRAPH '96)*, pp. 205–216, ACM Press, August 1996.

[30] Y. Arafa, K. Kamyab, S. Kshirsagar, N. Magnenat-Thalmann, A. Guye-Vuille, and D. Thalmann, "Avatar markup language," in *Proceedings of the 8th Eurographics Workshop on Virtual Environments (EWVE '02)*, pp. 109–118, Barcelona, Spain, May 2002.

[31] M. O. Riedl and R. M. Young, "From linear story generation to branching story graphs," *IEEE Computer Graphics and Applications*, vol. 26, no. 3, pp. 23–31, 2006.

[32] I. Baran and J. Popovi, "Automatic rigging and animation of 3D characters," in *Proceedings of the International ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '07)*, San Diego, Calif, USA, August 2007.