Appear in Proceedings of the Web3D 2003 Symposium, 2003

A Multi-User Virtual Environment System with Extensible Animations

Yi-Lin Liu
Computer Science Department,
National Chengchi University
Taipei, Taiwan 11623, ROC
g8901@cs.nccu.edu.tw

Tsai-Yen Li
Computer Science Department,
National Chengchi University
Taipei, Taiwan 11623, ROC
li@nccu.edu.tw

ABSTRACT

Multi-user virtual environment (MUVE) systems integrate the technologies of virtual reality and distributed system to allow users to interact with each other in a realistic virtual environment. However, due to the inflexibility in the current message protocol design, most of these systems can only allow a user to trigger canned animations in a sequential manner. In order to enable more flexible avatar animations, we propose to use XML as the language to design message protocol and extensible animations. We modified an open source MUVE system called VNet and replaced its message protocol with XML-based structures. We have also designed three animation functions to illustrate the extensibility of animation specification: avatar motion synchronization, flexible animation scripts, and integration of animations at different levels. Several animation examples have been designed to demonstrate how these functions can be used to enrich avatar animations in a MUVE system.

Categories and Subject Descriptors

H5 [**Information Interfaces and Presentation**]: H5.1 Multimedia Information Systems – *Animations, Artificial, augmented, and virtual realities* I.3 [**Computer Graphics**]: I.3.7 Three-Dimensional Graphics and Realism – *Animation, Virtual reality*.

Keywords

Multi-User Virtual Environments, Computer Animation, VNET, XML, VRML

1. INTRODUCTION

The development of virtual reality provides a new design direction for content presentation on the Internet in addition to texts and graphics. Multi-user virtual environment (MUVE) systems are an example application that uses the technologies of networking and virtual reality to provide realistic 3D visual effects and inter-person interactivity. However, due to the limited network bandwidth in the past, the applications of these systems were rather restricted. Nevertheless, the MUVE systems are now becoming prevalent because of the greatly improved network bandwidth and 3D technologies. Applications of these systems now are also increasing. Virtual art museums that display cultural relics and realistic space scenery for educational purposes are all examples of 3D applications.

In most virtual environment systems today, users use textual chatting, position changes and body animations to interact with other users. Most of them only use a simple message protocol of fixed format to exchange user information. The inflexibility of the message format makes it difficult to encode additional animation parameters in a message. Consequently, a user can only use a play-on-demand fashion to display canned animations triggered at run time. However, many interactions cannot be accomplished simply with sequential canned motions. For example, shaking hands with synchronized motion would be difficult to achieve in most MUVE systems today. Therefore, we would like to design a flexible message structure for these systems in order to improve the way that animations are specified for transmission and played on the client.

In recent years, XML (eXtensible Markup Language) has become the main data interchange format for Internet applications because of its rigorous definition and flexible structure. In this paper, we propose to use XML as the language to design the message protocol for MUVE systems. We utilize the standard XML structure with elements and attributes to describe extensible avatar animations. We have chosen a Java-based open-source MUVE system called VNet as our experimental platform and changed the original VRML Interchange Protocol (VIP) to an XML-based protocol. With this protocol, we have designed three avatar animation functions: *avatar motion synchronization*, *flexible animation scripts*, and *integration of animations at different levels*. With the improved MUVE, we hope to demonstrate that XML allows us to design message protocols in a more flexible way that enables extensible animations.

2. RELATED WORKS

Much research about MUVE systems has been proposed on system architecture and message protocols (Bouras and Tsiatsos 2000; Greenhalgh 2000; Huang et al. 1998), as well as applications in military simulation and education (Fellner and Hopp 1999). In terms of system architecture, one can classify the systems into two types: *client-server* and *peer-to-peer*. The client-server architecture is the most widely used one. For example, RING (Funk-Houser 1996) by UC Berkeley and AT&T Bell lab, Community Place (Lea et al. 19997) by the Computer Science lab and Architecture lab of SONY, Blaxxun Community Server (Blaxxun 2002) by Blaxxun, and ActiveWorlds (ActiveWorlds 2002) system by

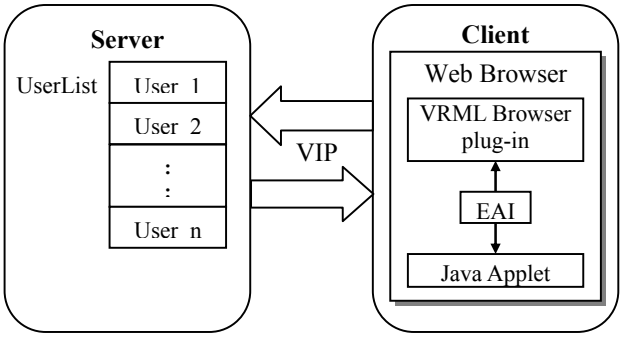


Figure 1. Original VNet system architecture

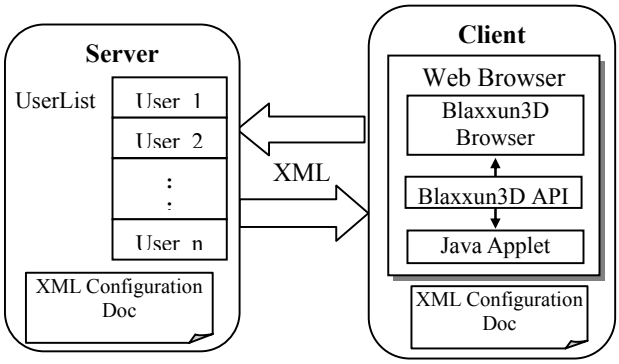


Figure 2. Proposed new VNet system architecture for extensible animations

ActiveWorlds are all examples that adopt the client-server architecture. VNet (VNet 1998) is another MUVE system with the client-server architecture that opens its source for cooperative development (Robinson et al. 2000). In this type of systems, since messages must be routed through the server, the server can easily become the bottleneck. Therefore, many researches try to address the problem of reducing the amount of data transmission by data filtering or Dead Reckoning techniques. In addition, there has been other research proposing the idea of using multiple servers to distribute the load on the server side. These researches all aim to increase the scalability of MUVE systems so that more users can be served at the same time.

The other approach, the peer-to-peer architecture, has been proposed to address the scalability problem. Examples with this approach include DIVE (Distributed Interactive Virtual Environment) (Frecon and Stenius 1998) by Institute of Computer Science in Swiss, MASSIVE (Model, Architecture, and System for Spatial Interaction in Virtual Environments) (Greenhalgh and Benford 1995) by Nottingham University, and NPSNet (Capps et al. 1998) by Naval Postgraduate School. Since no single server is in charge of scene management, the clients must be able to negotiate their roles and maintain the consistency of their data. Consequently, although the peer-to-peer architecture solves the bottleneck problem on the server, it increases the complexity of system deployment and design.

With recent computer development, more and more multimedia information such as audio, video, and animation are added into network applications. The network transmission protocols have also evolved to consider the characteristics of multimedia information in addition to traditional texts and graphics. For example, the Distributed Interactive Simulation Protocol (DIS) (Locke 1995) was developed for distributed simulation in military applications. The Interactive Sharing Transfer Protocol (ISTP) (Waters et al. 1997) has been built on top of HTTP, TCP, UDP, and RTP. The Distributed Worlds Transfer and communication Protocol (DWTP) (Broll 1998) was built on top of TCP and UDP to provide specialized functions for MUVE systems. VNet uses a simple application level protocol, called the VRML Interchange Protocol (VIP) (VIP 1998), for communications between avatars. Most of the protocols used by the above systems adopt a fixed format to compose messages, and one message can usually contain one type of data structure only. The clients must be able to decode this format in order to extract specific information in a message. However, with this type of hard-coded message format, applications cannot encode arbitrary information without referring to the design changes of message format.

The establishment of the VRML (VRML97 1997) standard has great influence on the development of 3D graphics on the network. Many recently developed MUVE systems have chosen VRML as the 3D format for the front-end 3D browser. However, the inconvenience of requiring plug-in installation and lack of good authoring and development tools hinders the prevalent development of 3D applications using this standard. On the other hand, XML (eXtensible Markup Language) (XML 2002), a markup language for defining other languages, has been widely accepted as the standard for data interchange on the network. Since XML does not have a fixed set of tags and grammar, the designer of an application has the freedom to create tag names and their relations in the context of the application. Due to its rigorous format definition and similarity with VRML, it is quite natural to merge the two standards to form the X3D (X3D 2002) standard. The browser based on the X3D standard can not only solve the aforementioned plug-in problems but also improve the previous application program interface, such as EAI (External Authoring Interface) (EAI 1997).

3. SYSTEM ARCHITECTURE

In this section, we first describe the basic architecture of our experimental platform, VNET, and the modifications that we have made to it. Then we will describe the pros and cons of using the original message transmission protocol: VRML Interchange Protocol (VIP) and how we have used XML to replace it.

3.1 Modifying VNet

As shown in Figure 1, VNet is a Java-based client-server MUVE system that adopts VIP as its message transmission protocol. VNet allows its clients to navigate through a virtual scene, chat with other users, and activate avatar animations. The server maintains a list of client status and broadcast new state updates to all clients. The clients must install a VRML browser to navigate through the environment and see other avatars. On the same web page, a Java applet for the VNet client interacts with the 3D browser through the External Authoring Interface (EAI).

In order to increase the portability and flexibility of VNet, we have made the following three changes to the original system. The proposed new system architecture is shown in Figure 2.

1. *3D browser*: We have used Blaxxun3D, an X3D implementation of the web3D browser developed by the Blaxxun Corporation, to replace earlier VRML browsers as the 3D display interface. Since the Blaxxsun3D browser is a pure Java-based implementation, a user does not need to install any additional

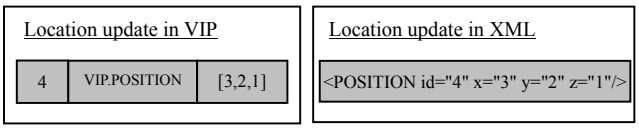


Figure 3. An example message in VIP and XML that moves a user (id 4) to location (1,2,3)

VRML browser in order to see 3D contents and therefore avoid the potential problem of system compatibility.

2. *Interface between the 3D browser and the client applet*: Since the 3D browser has been changed to Blaxxun3D, we also have to use the Blaxxun3D API to replace EAI as the programming interface between the 3D display applet and the VNet applet. Although the Blaxxun3D API is similar to EAI, it provides additional useful functions such as motion synchronization and therefore provides better compatibility with X3D.

3. *XML-based system configuration files*: Since XML was designed to describe the relations between data items, we have created two configuration setup files for the clients and the server. The server reads in the system-related parameters such as network ports and scene management data as the server starts. Similarly, a client obtains animation-related data such as library of canned motions or default scripts before it enters the first virtual environment.

3.2 Replace VIP with XML

A VIP message in VNet contains three components: *ID*, *Field*, and *Value*. ID is the user identification indicating who sends the message. Field indicates the message type, such as updating user location and having new user logging in. Value is the content of the Field, such as SFVector (three floating-point numbers) for location, and SFRotation (four floating-point numbers) for orientation. The main advantage of VIP is its simplicity for message parsing. However, the fact that a message can only contain one type of data at a time is over restricted.

In order to resolve the problem of fixed message format in VIP, we have used the element and attribute relations in the XML standard to encapsulate messages. For example, as shown in Figure 3, we have changed the message format from VIP to XML. We use the field name as the element tag, and the ID and value (x, y, z coordinates) as the attributes of the element. As for text-based messages, such as chat messages, we can simply enclose the text with an opening and closing tags and specify the receiving user ID as an attribute for the element tag. We have designed a full set of XML-based elements to replace the VIP protocol. Therefore, the VNet system can still provide all of its original functions such as chatting and animation for avatar motions after the message format has been changed to XML.

4. DESIGNING NEW ANIMATION FUNCTIONS

Replacing VIP is not the ultimate goal of changing the message format to XML. Instead, we would like to use the format to enable richer animations. In this section, we will use the XML-based messages to design three new animation functions: avatar motion synchronization, flexible animation scripts, and integration of animations at different levels.

4.1 Avatar Motion Synchronization

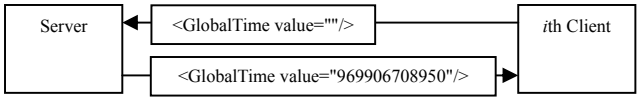


Figure 4. Message for getting server's time

```
<Synchronization
   requestId="1"
   receiveId="2"
   name="GiveFive"
   answer=""
/>
```

Figure 5. Message requesting synchronization

The objective of the first function is to solve a common problem that most MUVE systems have on synchronization of avatar interactions. When most of these systems receive an animation message, they can only play a canned animation immediately at their best efforts. Therefore, although most VRML browsers support real-time animations, the start time of an animation could be different on different machines simply because their network transmission times are not the same. Another user can only react to an animation message only after the animation has been played. Therefore, it is very difficult to perform interaction requiring cooperation with precise timing such as hand shaking.

To solve the synchronization problem, we must first be able to precisely control the start time of an animation on all clients. Our solution is to use a global time that all clients agree and know how to synchronize with it. We add a time attribute to an animation message to indicate the absolute global time that the animation should be played. This global time is set to the server time in our case. However, instead of resetting the clocks on the clients, each client only computes and remembers the difference between its local time and the global time according to the following three steps.

First, when the client i logs in the system or when it considers appropriate, a message requesting the global time will be sent to the server, as shown in Figure 4. Client i records the times when it sends the message and when it receives the reply message. The difference between the two times is defined as the two-way transmission time $\delta^i_{two\_way}$. Half of this time is roughly the one-way delay $\delta^i_{one\_way}$ for a message to travel between a client and the server. Second, assume that the global time in the message that client i receives is t_g. Then the real global time $t_{g'}$ when the client receives the message should be t_g plus the one-way delay $\delta^i_{one\_way}$. That is, $t_{g'} = t_g + \delta^i_{one\_way}$. Third, assume that the local time on the client machine i at the moment of receiving the message is t^i_l. The time difference d^i between the local time and the global time for the client i can then be defined as $d^i = t^i_l - t_{g'}$. When the client i receives a message requesting playing an animation at a global time, the client can compute the corresponding local time by adding d^i to the global time. Similarly, if a client would like to broadcast its animation to other clients, it subtracts d^i from its local time in order to obtain the global time for other clients to play the animation.

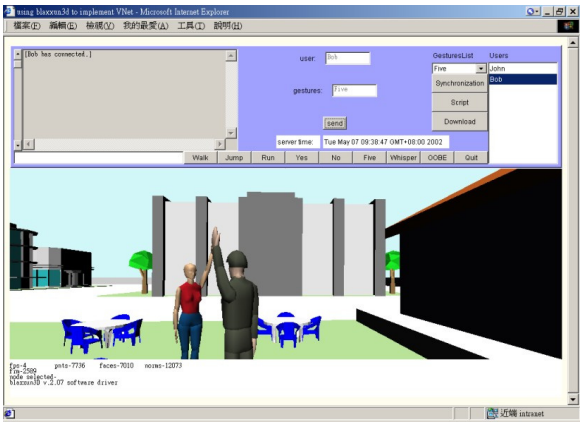
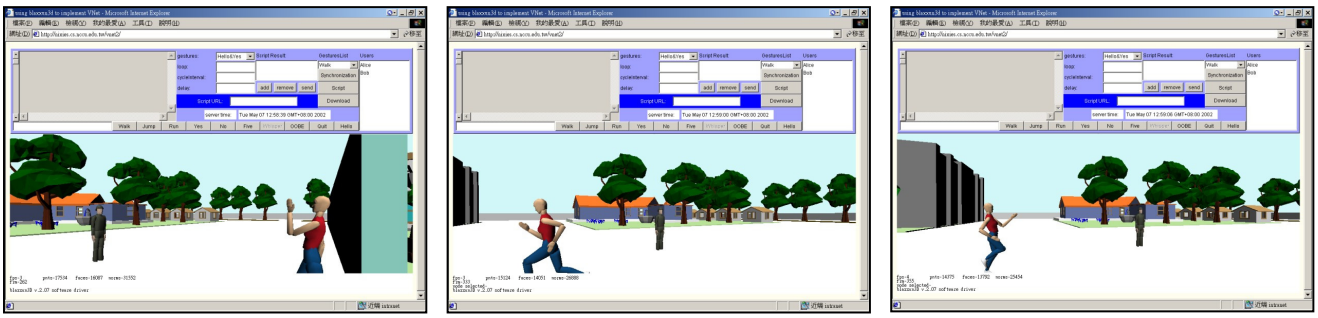


Figure 6. Animation of synchronized animation

```
<AvatarScript>                                          Animation sequence
  <AvatarAnimation name="HelloWalk" next="Run">   ❶
    <AnimationItem name="Walk"/>
    <AnimationItem name="Hello"/>
  </AvatarAnimation>
  <AvatarAnimation name="Run" next="ByeRun"/>     ❷
  <AvatarAnimation name="ByeRun">                 ❸
    <AnimationItem name="Run"/>
    <AnimationItem name="Hello"/>
  </AvatarAnimation>
</AvatarScript>
```

Figure 7. Animation script for an avatar



(a) (b) (c)

Figure 8. Playing animation based on a script (a) Hello and Walk (b) Run (c) Bye and Run

Figure 5 shows an example of message requesting a synchronized motion. In the example, we assume that the user of id 1 is requesting a synchronized "Give me five" motion from another user of id 2. A message window on user 2's machine will pop up to ask for an answer. If user 2 replies with an affirmative answer, the start times of the two avatars' motions will be set to the same global time. Thus, even a third client will be able to see the two avatars performing the synchronized animations, as shown in Figure 6.

Beside the global animation time attribute, we have also added several other attributes to an animation message. For example, an animation message can specify the number of replay loops, starting and ending points of a canned motion, and cycle time for animation speed. These are all examples of taking advantage of XML as the language to flexibly specify the parameters of an animation.

4.2 Flexible Animation Scripts

The second function is designed to show the flexibility of specifying an animation script in XML. We use the XML tree structure and the relations between elements to specify both sequential and parallel animations. When a client receives a message containing an animation script, it determines the playing time of each animation item according to the element name and its relations with other elements in the XML tree.

For example, assume that we are in a scenario of having a user run into some acquaintance while walking in an outdoor environment. He waves his hand and says hello while walking. Then something pops into his head, and he decides to run away. While running, he waves his hand to his friend again to say good-bye. We use the XML structure in Figure 7 to illustrate how the above sequence of animations can be specified. The script (<AvatarScript>) consists of three <AvatarAnimation> elements that are played sequentially. The name of each <AvatarAnimation> is specified by the user, and their playing order is determined by the "next" attribute. An <AvatarAnimation> without the "next" attribute would be the last animation to play. Each <AvatarAnimation> can contain multiple <AnimationItem> sub-elements that are played in parallel. The name of each <AnimationItem> corresponds to a canned animation that the client knows how to play from its startup script. Snapshots of the animation are shown in Figure 8.

In addition to using an XML tree to compose individual animations for an avatar in an <AnimationScript> element, we also have designed a <Script> element that can contain multiple <AnimationScript> elements for multiple avatars. With this type of element, the server can choose to encapsulate the animations for several clients into one message in order to reduce the number of transactions to the clients. This element becomes handy when we use a simulation program on a client to control a crowd of avatars. The design of this type of elements again demonstrates the extensibility of using XML as the language to express animations.

4.3 Integrating Animations of Different Levels

Instead of using canned motions that have been prepared in advance, we think it is important to be able to incorporate animations of different levels that could be generated at run time. In the above examples, we assume that the motions of an avatar are always canned motions (typically locomotion only) except for the

```
<AvatarAnimation name="MoveWalk" type="1">
  <action>move</action>
  <destination>JavaCafe.</destination>
</AvatarAnimation>
```

Figure 9. Specifying an animation by a simple high-level command

```
<AvatarAnimation name="MoveWalk" type="2">
  <AnimationItem name="Move">
    <PositionList>
      <POSITION type="SFVec3f" x="0.0" y="0.0"
                z="0.0" key="0.0"/>
      <POSITION type="SFVec3f" x="0.0" y="0.0"
                z="3.0" key="0.4"/>
      <POSITION type="SFVec3f" x="4.0" y="0.0"
                z="4.0" key="1.0"/>
    </PositionList>
  </AnimationItem>
  <AnimationItem name="Walk"/>
</AvatarAnimation>
```

Figure 10. Specifying an avatar animation with a given path

avatar's location. However, an animation command could be expressed at different levels of different complexity as long as both sides agree upon the format. For example, one can send a string "nodding" representing a high-level canned motion or send a sequence of low-level keyframes for the nodding motion. The visual effects resulting from both specifications should not be distinguishable by the clients. In this subsection, we would like to use the XML language to extend the animation function of VNet to various levels of specifications. We will use an example where an avatar walking to a Café to illustrate how high-level, mid-level, and low-level specifications can be used to describe the same animation.

First, an example of the high-level specification is shown in Figure 9. The example uses the <action> and <destination> elements to describe a command "move to Java Café." When a client receives a message like this, it determines the type of specification by the "type" attribute of the <AvatarAnimation> element. Then the animation program on the client prepares the animation by either looking up the canned motion from its motion library or generating the motion automatically with a motion planner. In this case, we use a shortest possible command to specify the motion but require the client to be able to understand and prepare the motions in details.

Second, we show the example with a mid-level specification of type two in Figure 10. In the specification, the <AvatarAnimation> element consists of two <AnimationItem>'s, called "Move" and "Walk", that will be played in parallel. The "Move" <AnimationItem> element consists of a sub-element called <PositionList>, which in turn contains a list of parameterized avatar positions (<POSITION>). We assume that the motion is equivalent to the first high-level command since the main difference is that the motion is prepared by the server instead of the clients. The server-generated list of positions will also lead the avatar to the same destination.

Third, we show the lowest level specification of type three for the same motion. Since we use a VRML browser to display the animation, the lowest level of specification would be similar to the

VRML：

```
DEF Path PositionInterpolator {
  key[ 0, 0.4, 1 ]
  keyValue [ 0 0 0, 0 0 3, 4 0 4 ]
}
ROUTE Path.value_changed TO Human_Root.set_translation
```

XML：

```
<AnimationNode name="PositionInterpolator" def_name="Path"
      route="Human_Root" field="translation">
  <key>[ 0, 0.4, 1 ]</key>
  <keyValue>[ 0 0 0, 0 0 3, 4 0 4 ]</keyValue>
</AnimationNode>
```

Figure 11. Specifying an avatar animation with low-level data similar to VRML

keyframe-based method used in the VRML browser. For example, to specify the path of an avatar, we can use the <AnimationNode> element in XML to encode the animation with the VRML mechanism, as shown in Figure 11. When a client receives a message like this, it will use the dynamic node creation feature of Blaxxun3D API to create and play the VRML animation node on the fly. One can specify the walk locomotion of an avatar in a similar fashion. Since the specification is at the lowest level, all the clients need to know is transform the XML format into the VRML format. Although the specification could be tedious and the size of the message could be large, it provides the most flexibility for the server to generate animations at run time.

The design of the above three levels of specifications demonstrates the extensibility of describing animations in XML. Figure 12 shows the animation results of using three levels of specifications for three different avatars. The results show similar visual effects and a client should not be able to distinguish which avatar is using which level of specification. With this function in the VNet system, we can allow the clients to have different animation libraries or different level of intelligence in generating their motions. The server will have the flexibility of choosing whatever specification producing the best result in term of efficiency and functionality.

5. CONCLUSION AND FUTURE WORK

In a MUVE system, the inflexibility of message format in the transmission protocol could impose a great constraint on how avatar animation can be specified. Consequently, most systems today can only display canned animations in a sequential manner. In this paper, we have proposed to use XML as the language to design an extensible message protocol for the VNet MUVE system. We have used XML based messages to replace the original VIP protocol. In addition, we have designed three functions to extend the animations that can be used in VNet. We believe that it is a trend to use XML as the language for delivering 3D contents on the network. The specification of X3D has demonstrated such a trend. In this paper, we have shown how it can also be used to specify extensible animations in a MUVE system.

The design of animation specification with XML elements for the three functions described in the paper is to illustrate the flexibility and extensibility of the XML structure. However, it is by no means complete in terms of how an animation can be expressed. For example, the current design of animation script can only contain one level of <AvatarAnimation> and one level of <Anima-

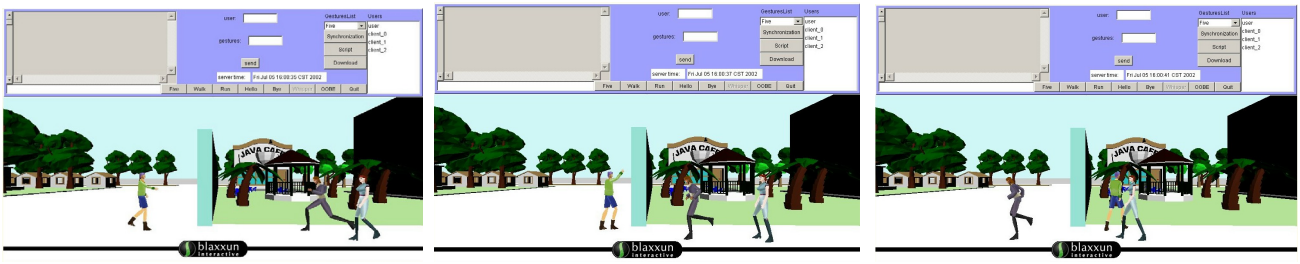


Figure 12. Snapshots of animations with three avatars using three different specifications

tionItem>. However, in some cases, it is more convenient to expression animations based on other existing ones. Therefore, allowing one <AnimationItem> to contain <AnimationItem> recursively could make animation specification easier. In addition, we only focus on the specification of avatar animations in this paper. However, actually we should be able to express general animations of any objects in XML. When the design of animation specification becomes more flexible, we should be able to allow different variety of designs to coexist. In this case, we can use the namespace mechanism of XML to solve the name conflict problem and generate different animations for the same name in different name spaces.

6. ACKNOWLEDGEMENT

This work was partially supported by grants from National Science Council under contract NSC 91-2213-E-004-005.

REFERENCES

Bouras, C., Tsiatsos T. 2000. pLVE: Suitable Network Protocol Supporting Multi-User Virtual Environments in Education, In *International Conference on Information and Communication Technologies for Education*, Vienna, Austria, 73-81.

Broll, W. 1998. DWTP-An Internet Protocol for Shared Virtual Environments. In *Proceedings International Symposium on the Virtual Reality Modeling Language 1998 (VRML'98)*. 49-56.

Capps, M., McGregor, D., Bruztman, D., and Zyda, M. 1998. Projects in VR: the Naval Postgraduate School's Moves curriculum. In *IEEE Computer Graphics and Applications,* 18, 3, 8-11.

Chiueh, T. C. 1997. Distributed Systems Support for Networked Games. In *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, 99-104.

Diehl, S. and Keller, J. 2000. VRML with Constraints. In *Proceedings of the Web3D-VRML 2000 fifth symposium on Virtual Reality Modeling Language*.

Fellner, D.W. and Hopp, A. 1999. VR-LAB - A Distributed Multi-User Environment for Educational Purposes and Presentations. In *Proceedings of the Fourth Symposium on the Virtual Reality Modeling Language*, Germany, 121-132.

Funkhouser, T. 1996. Network Topologies for Scalable Multi-User Virtual Environments. In *IEEE VRAIS'96*, 222-228.

Frecon, E. and Stenius, M. 1998. DIVE: a scalable network architecture for distributed virtual environments. In *Distributed Systems Engineering Journal (Special issue on Distributed Virtual Environments)*, 5, 3, 91-100.

Greenhalgh, C. and Benford, S. 1995. MASSIVE: a collaborative virtual environment for teleconferencing. In *ACM Trans. on CHI*, 2, 3, 239-261.

Greenhalgh, C. 2000. Implementing Multi-user Virtual Worlds: Ideologies and Issues. In *Proceedings of the Web3D-VRML 2000 fifth Symposium on Virtual Reality Modeling Language*, 149-154.

Huang, J. Y., Fang-Tsou, C. T., and Chang, J. L. 1998. A Multi-user 3D Web Browsing System. In *IEEE Internet Computing*, 2, 5, 70-79.

Lea, R., Honda, Y., and Matsuda, K. 1997. Virtual Society: Collaboration in 3D Space on the Internet. In *ACM CSCW'97, The Journal of Collaborative Computing*, 227-250.

Locke, J. 1995. An Introduction to the Internet Networking Environment and SIMNET/DIS. unpublished Master's thesis, Naval Postgraduate School.

Robinson, J., Stewart, J., and Labbe, I. 2000. MVIP-audio enabled multicast VNet. In *Proceedings of the Web3D-VRML 2000 fifth symposium on Virtual reality modeling language*, 103 - 109.

Waters, R. C., Anderson, D. B., and Schwenke, D. L. 1997. Design of the Interactive Sharing Transfer Protocol. In *Proceedings of Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 140-147.

ActiveWorlds, URL:<http://www.activeworlds.com>.

Blaxxun, URL:<http://www.blaxxun.com>.

EAI Specifiction. 1997. URL:<http://www .web3d.org/WorkingGroups/vrml-eai/>.

The VRML Interchange Protocol (VIP). 1998. URL:<http://www.csclub.uwaterloo.ca/u/sfwhite/vnet/VIP.html>.

VNet. 1998. URL:<http://www.csclub.uwaterloo.ca/u/sfwhite/vnet/>.

VRML97 spec. 1997. URL:<http://www.vrml.org>.

X3D 2002. URL:<http://www.web3d.org/x3d/>.

XML 2002. URL:<http://www.w3.org/XML>.