IMNET: An Experimental Testbed for Extensible Multi-user Virtual Environment Systems

Tsai-Yen Li, Mao-Yung Liao, and Pai-Cheng Tao

Computer Science Department, National Chengchi University, Taipei, Taiwan
{li, g9105, g9310}@cs.nccu.edu.tw

Abstract. Multi-user virtual environment (MUVE) systems enable virtual participation in many applications. A MUVE usually is a complex system requiring technologies from 3D graphics and network communication. However, most current systems are designed to realize specific application contents and usually lack system extensibility. In this paper, we propose an extensible architecture for a client-server based MUVE system called IMNET. This XML-based MUVE system allows function modules to be flexibly plugged into the system such that network or user interface experiments can be easily incorporated. We will use two examples to illustrate how to flexibly change the system configurations on the server and client sides to enhance system functions or to perform experiments. We believe that such an experimental test-bed will enable a wider range of researches to be carried out in a more efficient way.

1 Introduction

A multi-user virtual environment (MUVE) system is a system allowing many users to share the same 3D virtual world through the network and participate in the activities in the world as avatars. It allows a user to interact with other users or the environment via textual or visual communications. A snapshot of the user interface in a virtual environment is shown in Fig. 1. The feature of not being constrained by physical existence allows such a system to have a great potential value in applications that cannot be easily realized in the real world. For example, a 3D role-playing game allows its users to act as a fictional characters in an ancient world. A MUVE can also be adopted to simulate military activities. In addition, many examples have demonstrated that it can also be used to visualize or perform scientific experiments that cannot be easily explained by texts and figures [3].

Many MUVE systems have been proposed in the literature. In early years, most systems were developed for research purposes. However, in recent years, one can see more commercial systems being designed to host such a virtual environment for general purposes or for special purposes such as on-line games. Designing a MUVE is a complex task requiring multi-discipline trainings involving networking and 3D technologies. Most MUVE systems are packaged as a standalone application or a program module that can be embedded in a web page. Although some of them may have external application programming interface (API) for integration with other programs [12], most of them cannot be extended at design time or configured at run time. In this paper, we propose an experimental MUVE test-bed, called *IMNet* (*Intelligent Media*

O. Gervasi et al. (Eds.): ICCSA 2005, LNCS 3480, pp. 957 – 966, 2005.

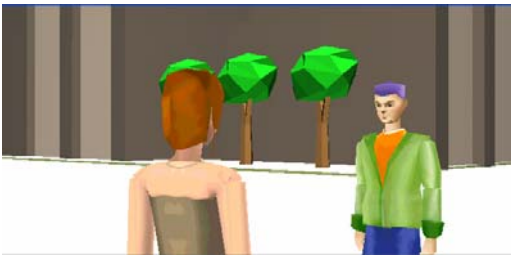

Fig. 1. An example dialog scene in a virtual environment

Network), that is designed to be extensible for incorporating other function modules such as message filters or user interface components. IMNet adopts a client-server architecture and uses XML as the base language for server and client configurations as well as the message protocols for MUVE. We will demonstrate the extensibility of our system by two examples incorporated into the server and client programs, respectively.

The rest of the paper is organized as follows. We will review the work pertaining to MUVE systems. In the third section, we will describe the proposed extensible system architecture for the server and client programs. We will describe the message protocol and its encoding in the fourth section. Two examples will then be given to illustrate the functions of the experimental test-bed. Finally, we will conclude the paper with some future extensions.

2 Related Work

The MUVE related research proposed in the literature has various aspects. Some of them focus on system architecture and message protocols [1][5][6] while others focus on applications such as in military simulation and education [3][2]. In terms of system architecture, most systems fall into two types: *client-server* and *peer-to-peer*. The client-server architecture is the most widely used one. For example, RING [4] by UC Berkeley and AT&T Bell lab, Community Place [7] by the Computer Science laboratory and Architecture laboratory of SONY, Blaxxun Community Server [13] by Blaxxun, and ActiveWorlds [12] system by ActiveWorlds are all examples that adopt a client-server architecture. VNet is another MUVE system with a client-server architecture that opens its source for cooperative development [10]. 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, some research proposes the idea of using multiple servers to distribute the load on the server side [11]. This type of research aims to increase the scalability of a MUVE system such that more users can be served at the same time. The experiments in much of this research are done by modifying a specific MUVE system, and the implementation cannot be easily ported to other systems.

In addition to the communication issues, standards for 3D animation and display on the client side also attract many attentions. Many recent MUVE systems use open

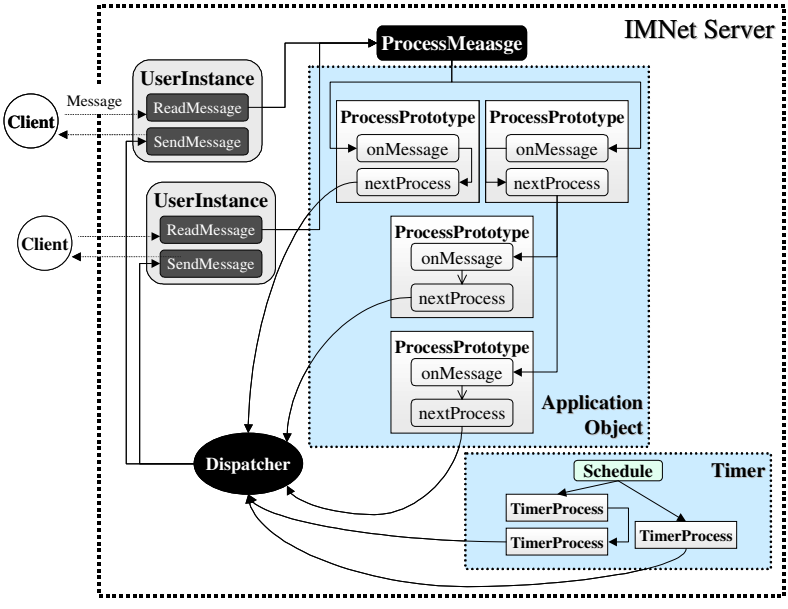

Fig. 2. System architecture of IMServer with the pluggable message processing mechanism

standards such as VRML [15] to model the geometry and animation of a virtual scene while the other use proprietary formats. The 3D display program is usually packaged as a 3D browser that can be embedded in a web page. The browser could be stand-alone or integrated with other external programs such as a Java Applet to provide application-specific functions [13]. Due to the extensibility of XML (eXensible Markup Language), the VRML standard is migrating into the XML-based X3D language [18]. However, most MUVE systems use a fixed proprietary format as the application protocol for message passing [10]. [9] is an example that attempts to change the message format of VNet to an XML-based protocol. Although the system designer can easily design new tags to enrich the functions of a MUVE, the programs on the client and server sides needs to be modified to accommodate the changes. The system is not designed to incorporate plug-in modules that are specified at design time or even at run time. In addition, although the protocol is more extensible, the size of a same message could be larger if raw XML strings are used for transmission.

3 System Description

In this section, we will describe the system architecture of the server and client programs in the IMNet virtual environment system. IMNet is a client-server based MUVE system adopting XAML (eXensible Animation Modeling Language)[8] as the language for 3D display and animation. The server program is called *IMServer* while the client is called *IMClient*. XAML is an animation scripting language that is designed to specify animations in a range of abstractions. For example, it can be used to

specify low-level joint values as in VRML. It can also be used to specify high-level goal-oriented motions such as "Move to Café" as long as the animation engine knows how to interpret the script and generate the animation.

```
<serverConfig>
   <processors>
      <processor class="example.processorA">
         <processor class="example.processorC" />
         <processor class="example.processorD" />
      </processor>
      <processor class="example.processorB" />
      <timerprocessor class="exampleTestTimer" delay="5000">
         <processor class="example.processorE" />
      </timerprocessor>
      <timerprocessor class="exampleTestTimer" delay="2000" />
   </processors>
</serverConfig>
```

Fig. 3. Example of server configuration on message processing structure

3.1 Server System Architecture

According to [7], a MUVE system consists of four modules, Client, Server, Application Object, and Server Client Protocol. The Application Object (AO) module is responsible for interpreting the messages and managing the application contexts (for example, virtual shopping mall). Data filtering routines such as dead reckoning algorithms can also be implemented in the AO module to improve the performance of the server by filtering out unnecessary information for the clients.

The system architecture of IMServer including the AO module is shown in Fig. 2. When a client logs into the system, a `UserInstance` is created on the server to take care of the message input and output for the client. All messages are sent to the `ProcessMessage` routine for data processing in the AO module. Each data processing unit in the AO module is called `ProcessPrototype`. All `ProcessPrototype`'s in the AO module are organized as a tree structure to process the message data in parallel or in sequence. The `onMessage` method of the first `ProcessPrototype` in each tree branch is called to process the message data and decide if it will pass the data to the next `ProcessPrototype` or simply filter them out. Each of the leave `ProcessPrototype`'s in a tree may generate messages to the dispatcher for distribution to other clients. In addition to being driven by the incoming message events, the server can also produce messages voluntarily through the timer service. The processing units of the service are also organized in a tree structure such that they can work together in parallel or in sequence.

With best extensibility in mind, we have designed a mechanism to set up the above processing tree at run time on the server side. This mechanism is described as an XML configuration file, as the example shown in Fig. 3. The java class is specified in the "class" attribute of each process. In this example, process A and B are the start of

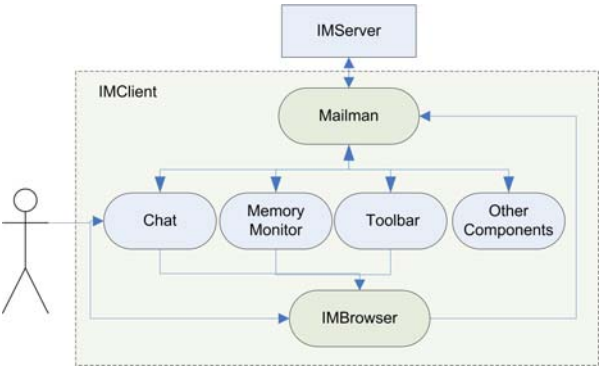

Fig. 4. System architecure of IMClient

```
<imclient>
 <components>
    <component class="Mailman" name="mailman"/>
    <component class="SpChat" name="chat"/>
    <component class="SpActionButton" name="actionButton"/>
    <component class="SpMemoryMonitor" name="memoryMonitor"/>
    <component class="SpIMBro" name="browser"/>
 </components>
 <eventDispatcher name="mailman">
    <connect ip="127.0.0.1" port="62266"/>
    <eventlistener name="chat"/> ...
 </eventDispatcher>
 <toolbar>
    <button name="memoryMonitor"/> ...
 </toolbar>
 <toolbar>
    <buttonContainer name="actionButton">
       <xamlButton name="bow" text="Bow"
file="Behavior/Bow.xml"/>
       ...
    </buttonContainer>
 </toolbar>
 <panels>
    <panel name="browser"/>
    <panel name="chat"/>
 </panels>
</imclient>
```

Fig. 5. An example of configuration file for IMClient

the two branch processes. Process A first filters the messages and pass them to process C and D s equentially. Since both processes B and D are the last process in their braches, they may produce messages that will be distributed to other clients. In addition, independent timer processors can be evoked periodically according to the specified delays. Since the processing routines' classes are organized and bound at run time, experiments can be done easily by specifying appropriate filtering or processing routines in the configuration file without recompiling the server's code.

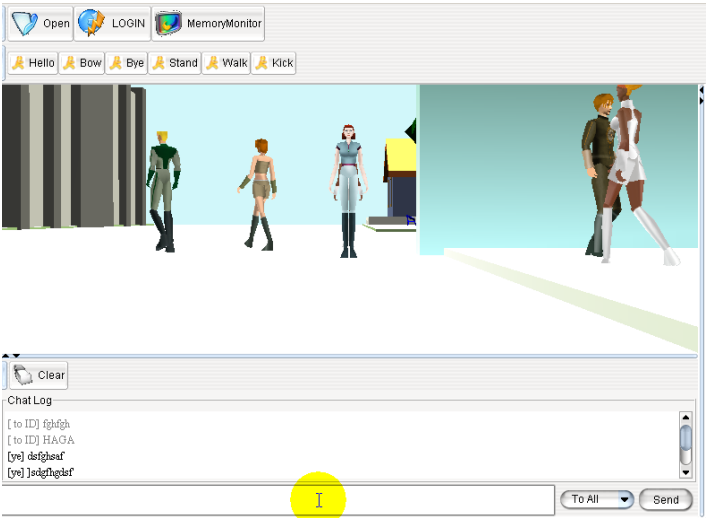

Fig. 6. A snapshot of IMClient user interface for the configuration in Fig. 5

3.2 Client System Architecture

The client side program of IMNet is called IMClient. The system architecture of IM-Client is depicted in Fig. 4. The program consists of two major components: Mailman (communication module) and IMBrowser (3D animation engine), and other GUI components such as textual chat and action buttons. The program is updated according to two types of events: messages from the server and actions from the user. A message from the server is first processed by the Mailman module and passed to all other interested modules. Corresponding components of the screen will be updated according to the type of the message such as a movement or a chat message. The user can also create events, such as entering chat messages or clicking on action buttons, to be sent to the server via the Mailman module.

A main feature of IMClient is that the components comprising the program can be configured at run time. The program is set up by loading a configuration file at initialization such as the one shown in Fig. 5. In this file, each class module is defined as a named component which may or may not contain a GUI widget. The relations among these components are set up according to the event dispatcher and listener model. The latter part of this file describes how the components are connected to the GUI widgets. For example, both IMBrowser and Chat implement a panel widget to be arranged in the client window as shown in Fig 6. Two types of toolbars are also used in the client window: static toolbar and dynamic toolbar. A static toolbar contains buttons that must be initialized at start-up time while dynamic toolbar allow buttons to be created and inserted at a later time. For example, an action button of an avatar for a canned motion in a MUVE can be downloaded from the server as long as the canned motion is described by a XAML script.

```
<IMNet from="userA" to "userB">
   <Chat> See you later. </Chat> <!—textual>
   <AnimItem> <!- XAML script>
      <AnimImport src="Bye"/>
   </AnimItem>
</IMNet>
```

Fig. 7. An example message in IMNet with textual and animation contexts

4 Message Protocol

4.1 IMNet Message Protocol

The message protocol used in IMNet adopts XAML as the base animation scripting language. The protocol needs to deliver messages containing information such as user login events, movements, and animation. Since complex and extensible animations can be embedded in an XAML script, the remaining message types for the virtual environment application can be kept minimal. In the current design, the additional tags include the following: `<IMNet>`, `<Chat>`, `<Login>`, `<Logout>`, and `<UserMove>`. In Fig. 7, we show an example message about a user A whispering to user B while performing an animation described in an XAML script, stored in a separated file. In addition, a login message uses the format of `<Login id="userC" url="wrl/avatar_03.wrl">` while a user movement message uses the format of `<UserMove x="10" y="20">`. The latter message can also be described in an XAML script, but we make it a standalone message to optimize this type of frequently used actions.

4.2 Message Encoding

Although messages in the XML format have the advantage of being extensible, they also have the drawback of being large in size. The problem gets worse for a MUVE system when the animation gets more low-level and complex. Similar problems also arise in the WAP (Wireless Application Protocol) [16] application. The WAP development community proposed an encoding method called WBXML (WAP Binary XML) [17] to convert XML string into a concise binary format. We have also adopted such an encoding method to deliver IMNet messages. However, instead of converting an XML string to WBXML, we generate WBXML directly from an internal DOM (Document Object Model) for efficiency.

We have done experiments to compare the encoding and decoding performance of different methods as well as with the original XML format. The experimental data, as shown in Fig. 5, were measured on a personal computer with an AMD XP2500 processor. Note that the WBXML encoding method outperforms the Java serialization method and the common ZIP compression method in encoding time as well as decoding time. However, the string size after the WBXML encoding is 2.5 times larger than the one with ZIP compression on average although the size has been reduced by 4.2 times on average compared to the Java serialization method.

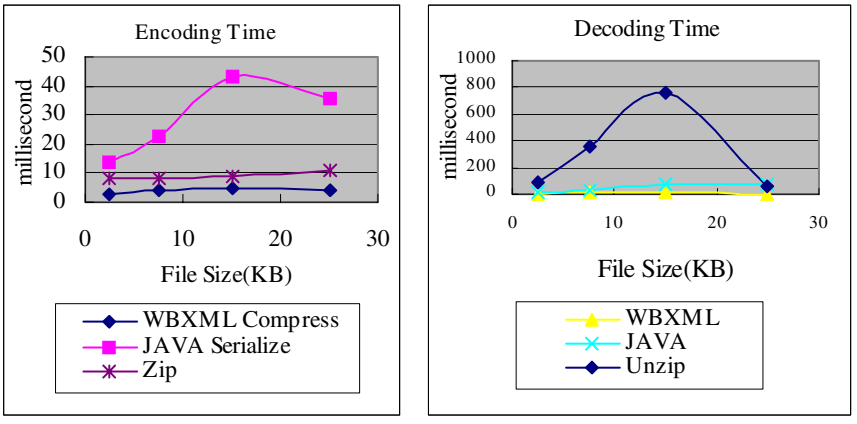

Fig. 8. Comparisons of encoding and decoding performances

5 Examples of System Extensibility

5.1 User-Centric Throughput Adjustment Experiments

For a client-server based MUVE system like IMNET, the server is commonly considered as a bottleneck for message exchanges. Therefore, much research has proposed to use the idea of data filtering to reduce the amount of traffic that needs to be transmitted across the network. The decision is usually made by some intelligent modules such as view culling and dead reckoning on the server side to filter out unnecessary information according to each client's configuration. In fact, each client's ability in display and network I/O may vary greatly, and a uniform policy is not going to fit every clients need and may waste the server's resources in sending out unnecessary messages that the clients cannot digest. Other factors such as message types and user activities may also imply the demands for customized filtering policies according to the user's model. The server should also adjust its message update frequency according to its own CPU and network I/O performance. We call a server with this type of capability a server that can perform user-centric throughput adjustment.

Fig. 9 shows the system configuration, similar to Fig. 2, for this experiment. The performance monitoring module subscribes the incoming messages and monitors the CPU and I/O performance of the server. These data are maintained as the system states for the server and clients. The other branch of the message flow starts from the dead reckoning module, which filters out messages for the clients keeping the expected moving direction. The filtered messages will be passed to the throughput control module which determines whether the messages should be sent to each specific client or not according to their system states and the server's current loading. The filtered messages are sent out to the clients via the dispatcher module. Note that setting up a system experiment like this does not require the designer to recompile the program since the message flows are set up at run time according to a system configuration file similar to the one shown in Fig. 3. This feature allows the designer to insert or remove different experimental modules and treat the system as a flexible

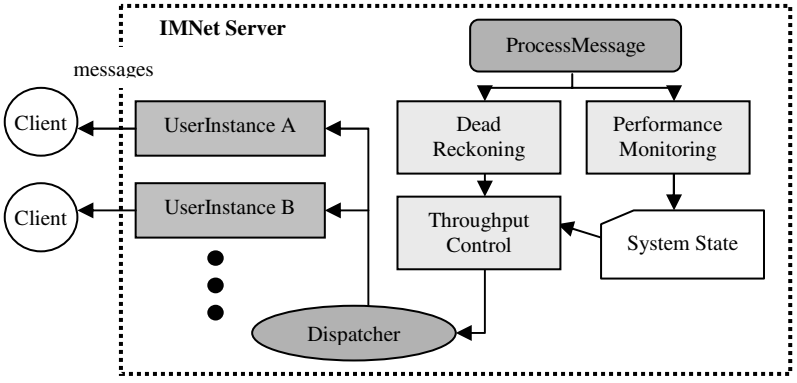

Fig. 9. An example of server configuration for the adjustable throughput control experiment

or remove different experimental modules and treat the system as a flexible experimental test-bed.

5.2 Extension to Voice-Enabled User Interface

One can also extend the functions of IMClient by specifying appropriate components in the configuration file. We will use a voice-enabled user interface as an example to illustrate how to add new system functions and user interface modules into the system. The voice-enabled user interface is based on a research aiming to incorporate voice dialogs in MUVE [8]. The system allows two avatars in the same scene to talk to each other via voice dialogs and embedded animations while allowing a third user to observe the dialog. The protocol that the system has used is called XAML-V since it acts as a plug-in extending the XAML animation scripting language. Assume that this new module is called the VUI module. It implements a text panel for displaying voice dialogs and subscribes to the incoming messages via the Mailman component. This new module can be hooked up to IMClient with ease by modifying the XML-based configuration file in Fig. 5.

6 Conclusions

In this paper we have described an experimental testbed for MUVE. This system adopts an XML-based protocol that allows an extensible animation scripting language to be efficiently embedded in the messages. The extensibility of the system is also shown in the system configurability of the server and clients by two examples. We believe that the extensibility of this system will enable more research to be conducted in a more efficient way.

Acknowledgement

This work was partially supported by a grant from National Science Council under a contract NSC 93-2213-E-004-001.

References

1. Bouras, C., Tsiatsos, T.: pLVE: Suitable Network Protocol Supporting Multi-User Virtual Environments in Education. In: International Conference on Information and Communication Technologies for Education, Vienna, Austria (2000) 73-81
2. Elliott, C, Lester, J. C., Rickel, J.: Integrating affective computing into animated tutoring agents. In: Proceedings of IJCAI '97 workshop on Intelligent Interface Agents (1997)
3. Fellner, D.W., Hopp, A.: 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 (1999) 121-132
4. Funkhouser, T.: Network Topologies for Scalable Multi-User Virtual Environments. In: Proceedings of IEEE VRAIS'96 (1996) 222-228
5. Greenhalgh, C.: Implementing Multi-user Virtual Worlds: Ideologies and Issues. In: Proceedings of the Web3D-VRML 2000 fifth Symposium on Virtual Reality Modeling Language (2000) 149-154
6. Huang, J. Y., Fang-Tsou, C. T., and Chang, J. L: A Multi-user 3D Web Browsing System. In IEEE Internet Computing, Vol. 2, 5, (1998) 70-79
7. Honda, Y., Matsuda,, K., Rekimoto, J., Lea, R.: Virtual Society: extending the WWW to support a multi-user interactive shared 3D environment. In: Proceedings of the First Symposium on Virtual Reality Mmodeling Language (1995) 109-166
8. Li, T.Y., Liao, M.Y., Liao, J.F.: An Extensible Scripting Language for Interactive Animation in a Speech-Enabled Virtual Environment. In: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME2004), Taipei, Taiwan (2004)
9. Liu, Y.L., Li, T.Y.: A Multi-User Virtual Environment System with Extensible Animations. In: Proceedings of the Web3D 2003 Symposium (2003)
10. Robinson, J., Stewart, J., and Labbe, I.: MVIP-audio enabled multicast VNet. In: Proceedings of the Web3D-VRML 2000 Fifth Symposium on Virtual Reality Modeling Language (2000) 103-109
11. Smed, J, Kaukoranta, T., Hakonen, H.: A Review on Networking and Multiplayer Computer Games. In :Technical Report 454, Turku Centre for Computer Science (2002)
12. ActiveWorlds, http://www.activeworlds.com
13. Blaxxun, http://www.blaxxun.com/
14. VoiceXML, http://www.w3.org/TR/voicexml20/
15. VRML, http://www.web3d.org/x3d/specifications/vrml/vrml97/
16. WAP, http://www.wapforum.org/
17. WBXML, http://www.w3c.org/TR/wbxml/
18. X3D, http:///www.web3d.org/x3d