# Realising voice dialogue management in a collaborative virtual environment

Chun-Feng Liao* and Tsai-Yen Li

Department of Computer Science,
National Chengchi University,
64, Sec. 2, Zhih-Nan Road,
Taipei, Taiwan 116, ROC
E-mail: try@nccu.edu.tw
E-mail: li@nccu.edu.tw
*Corresponding author

**Abstract:** The applications of 3D-virtual environments and Voice User Interface (VUI) on personal computers have received significant attention in recent years. Since speech is the most natural way of communication, incorporating VUI into virtual environments can greatly enhance user interaction and immersiveness. Although there have been many researches addressing the issue of integrating VUI and 3D-virtual environment, most of the proposed solutions do not provide an effective mechanism for multiuser dialogue management. The objective of this research is on providing a solution for integrated dialogue management and realising such a mechanism in a collaborative Multi User Virtual Environment (MUVE). We have designed a dialogue scripting language called eXtensible Animation Markup Language – Voice Extension (XAML-V), based on the VoiceXML standard, to address the issues of synchronisation with animation and dialogue management for multiuser interaction. We have also realised such a language on a MUVE to evaluate the effectiveness of this design.

**Keywords:** voice user interface; VUI; VoiceXML; dialogue management; multi-user virtual environment; MUVE.

**Reference** to this paper should be made as follows: Liao, C-F. and Li, T-Y. (2007) 'Realising voice dialogue management in a collaborative virtual environment', *Int. J. Computer Applications in Technology*, Vol. 29, No. 1, pp.37–44.

**Biographical notes:** Chun-Feng Liao received his BS and MS from National Chengchi University in 1998 and 2004, respectively. He is currently a PhD student in the Computer Science and Information Engineering Department at National Taiwan University. His research interests include Intelligent Systems and Context-Aware Middleware.

Tsai-Yen Li received his BS in 1986 from National Taiwan University, and MS and PhD in 1992 and 1995, respectively, from Stanford University. He is currently a Professor in the Computer Science Department of National Chengchi University in Taiwan. His research interests include Computer Animation, Intelligent User Interface, Motion Planning, Virtual Environment, and Artificial Life.

## 1 Introduction

Due to the rapid development of graphics hardware and software, virtual reality that used to run on high-end graphics workstations can now be experienced on desktop computers. Among the potential applications, *Multi-User Virtual Environment* (*MUVE*) (or called *Collaborative Virtual Environment (CVE)*) is one that allows many users to share their experiences in a 3D-virtual environment (Matijasevic, 1997). The nature of this type of system requires tight integration of multimedia (especially 3D graphics) and distributed system technologies. An example application of this type of environment is the prevalent 3D-online games that have received significant attentions in recent years. Other applications on military, entertainment, education, etc. are also emerging (Apaydin, 2002; Wauchope et al., 2003).

Most MUVE systems today, such as DIVE (Frecon and Stenius, 1998), MASSIVE (Greenhalgh and Benford, 1995), Blaxxun (http://www.blaxxun.com) and ActiveWorld (http://www.activeworlds.com), adopt a multimodal user interface at least consisting of 3D-navigation and textual chatting. However, few of them have incorporated Voice User Interface (VUI), the most natural way of communication for human beings, into their systems, despite the recent advances in speech-related technologies. We think the main reasons are twofold. Firstly, there exists no effective dialogue management mechanism for multiple users across the network in general. Most of the voice applications today are simple two-party applications focusing on the voice dialogues between a human and a machine playing the role of the other party. Secondly, there is no flexible way to integrate dialogue specifications seamlessly into a computer-generated animation in the current MUVE systems.

In this paper, we propose a dialogue management mechanism that enables VUI in a MUVE. The mechanism uses a protocol to let two avatars, representing either human beings or machines, establish a dialogue connection and allow other avatars in the virtual world to observe the progress of the dialogue. The protocol is realised with XML-based documents while the dialogue itself is a form based on VoiceXML (http://www.w3.org/Voice/). Due to the extensibility of XML, this dialogue management mechanism is seamlessly integrated into a MUVE system called Intelligent Media Network (IMNet) (Li et al., 2005) that adopts eXtensible Animation Markup Language (XAML) (Li et al., 2005) as the underlying animation scripting language. The VUI is described with a language called XAML – Voice Extension (XAML-V) and embedded in an XAML script as a plug-in which can in turn trigger additional animation scripts inside the dialogue.

In Section 2, we will briefly review the related work in MUVE and dialogue management. In Section 3, we will describe the requirements of enabling voice dialogues in a MUVE. The design of XAML-V for realising such a voice interface will then be presented in the following section. In Section 5, we will describe some implementation issues and illustrate our design with an example dialogue between two users and observed by other users. Finally, we will conclude this paper with some future research directions.

## 2 Related work

### *2.1 Scripting languages for MUVE*

The application protocol for delivering multimodal contents, such as 3D-animation and textual chat, has also been an active research topic. A recent trend is on designing an XML-based animation scripting language for describing the activities in a virtual environment. For example, Avatar Markup Language (AML) (Kshirsagar et al., 2002) focuses on facial expression but only provides limited functions for altering a canned motion. STEP is another XML-based scripting language that emphasises on its logical reasoning ability (Huang et al., 2002). XAML is also an XML-based animation scripting language featuring its extensibility in modelling animations with various levels of controls and allowing other external modules to be incorporated (Li et al., 2004). In this work, we have chosen to extend XAML to incorporate a mechanism for voice dialogue management in the IMNet (Li et al., 2005) system.

### *2.2 Dialogue management*

The researches for voice technologies, such as speech synthesis, speech recognition and their applications have made significant progresses in recent years. International standards such as VoiceXML are emerging as the de facto for dialogue-based applications. Most of these designs aim to provide a VUI to a user by downloading a dialogue script from a document server. However, since two-way communications between a human being and a computer are usually the basic assumption for designing such a language, it cannot be directly applied to a MUVE system without modifications.

Since multiple clients could be interacting with each other at the same time in MUVE, the dialogue session management mechanism is essential to prevent collisions of dialogue requests and to ensure the quality of dialogue. Among the dialogue session management protocols, H.323 (Packet-based multimedia communications systems, http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.323.) or Session Initiation Protocol (SIP) (http://www.ietf.org/html.charters/sip-charter.html/) are two of the most important connection management mechanisms today. H.323 was designed for teleconferencing while SIP was designed for voice applications based on Voice Over IP (VOIP) technologies. The ideas of the session management control protocol used in our system are similar to these two standards but the message contents, protocol format and hosting environments are all very different.

### *2.3 Integration of voice dialogue and virtual environment*

Galatea (Sagayama et al., 2003) is an Anthropomorphic Spoken Dialogue Agent (ASDA) platform that makes use of the dialogue model of VoiceXML. GalateaDM is one of the modules on the platform for dialogue control. It extends VoiceXML to incorporate animation descriptions such as facial expression scripts. However, it was not designed to be used in a MUVE and therefore has not taken multiparty dialogue management into account.

In Nyberg et al. (2002), the authors argue that the form-filling mechanism in VoiceXML is insufficient for expressing state transitions in an advanced dialogue. Therefore, a language called DialogXML is designed to express a more complex dialogue. A dialogue manager is also designed to translate the scripts in this language into VoiceXML scripts at run time. Although, a high-level language like DialogXML can facilitate designing a dialogue with the low-level VoiceXML language, it also comes with disadvantages. For example, translating the scripts at run time in a MUVE may affect the system performance and using state diagram to design a dialogue without a good authoring tool may not be intuitive to regular users.

Due to the complexity of system design, most voice interfaces and virtual environment systems existing today were developed independently. Therefore, some efforts have to be made to facilitate the integration. For example, Cernak and Sannier (2002) has used the CSLU toolkit to facilitate the integration. The voice interface can communicate with the virtual environment through the CORBA mechanism. However, as the authors pointed out, the efficiency of the overall system suffers under such a distributed environment. In Descamps et al. (2001), the authors use eXtensible Stylesheet Language Template (XSLT)[1] to translate Multimodal Presentation Markup Language for VRML (MPML-VR) into Javascript and

VRML. The voice interface module communicates with the VRML browser via the External Authoring Interface (EAI) provided by the VRML browser. A main disadvantage of such an integration mechanism is that most EAI available today can only support a specific version of Java virtual machine and therefore limit the extensibility of such an integration solution.

## 3 Dialogue management in MUVE

VoiceXML was originally designed for dialogues between human being and system in a telephony environment. A human user interacts with the system by retrieving a sequence of dialogue forms from a document server just as we do in a typical session of a web application. In such an environment, there are at most two participants in a dialogue session. However, in a typical MUVE, the number of avatars in a scene is usually much larger. In a dialogue session, two avatars are the active subjects while the other avatars act as observers. In order to clarify the roles of the avatars in a typical MUVE, we have adopted the following notations.

Subjects: Avatars in a dialogue.

Observers: Avatars not in a dialogue.

U: Avatars controlled by human being.

S: Avatars controlled by system.

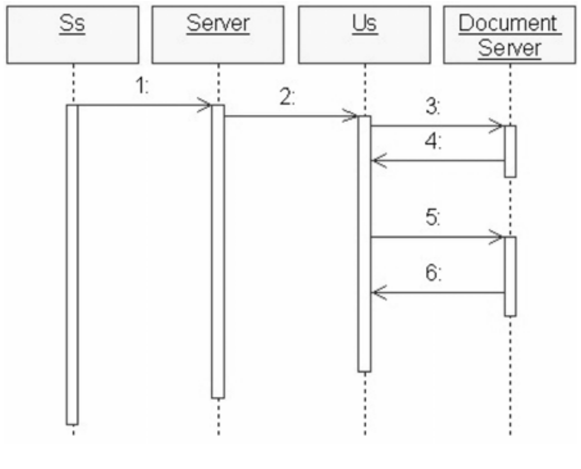
Suffix s: Subject avatars.

Suffix i (i = 1, 2, 3, …): Observer avatars.

For example, $U_s$ denotes an avatar in dialogue controlled by a human user.

If we adopt the session management model of a typical VoiceXML session between two avatars controlled by a human ($U_s$) and a machine ($S_s$), the dialogue may actually happen between $U_s$ and the document server as shown in Figure 1. After the dialogue is initialised, $S_s$ sends its dialogue script's URL to $U_s$ (Figure 1, steps 1–2), and then $U_s$ fetches the script according to this URL from the document server and collects inputs from the user. A new script is then fetched based on the user's response (Figure 1, steps 3–6).

**Figure 1** Sequence diagram of applying the VoiceXML session management model to a MUVE

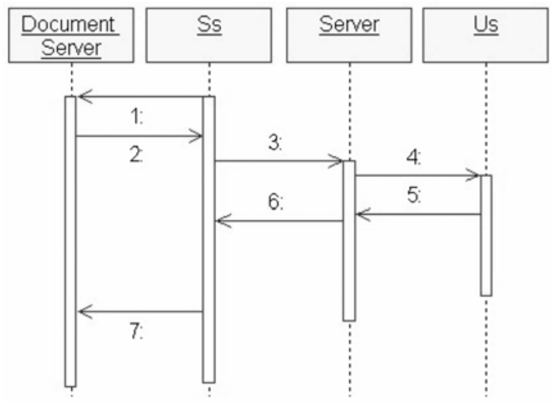


When applying the VoiceXML dialogue model to a MUVE as described above, one may encounter several problems. Firstly, although $S_s$ is in a dialogue session with $U_s$, $S_s$ is not aware of the dialogue status after sending out the URL of the first dialogue script. If some network failures occur during the dialogue or $U_s$ deliberately stops the dialogue, $S_s$ will not be notified and updated. Secondly, without a mechanism to maintain the dialogue status, $S_s$ may be talking to two or more avatars simultaneously or showing a mixed and confused animation to a wrong target. Therefore, we have proposed several mechanisms as described below to enhance the original session management model.

### *3.1 Proxy request*

In order to make $S_s$ be aware of the dialogue status when talking with $U_s$, we propose to use a proxy-request mechanism as for a proxy server on WWW. In the enhanced model, all requests of dialogue scripts must pass through $S_s$ as shown in Figure 2.

**Figure 2** The sequence diagram of adopting the proxy-request mechanism in a dialogue



In Figure 2, $S_s$ is responsible for fetching the dialogue scripts that are requested by $U_s$ (Figure 2, steps 1–2) and then send these scripts to $U_s$ (Figure 2, steps 3–4). To continue the dialogue, $U_s$ may request the next dialogue script with some feed back information (e.g. answering some question proposed by $S_s$). The information will be sent back to $S_s$, then $S_s$ fetches the scripts again and continues the interaction loop until the dialogue ends.

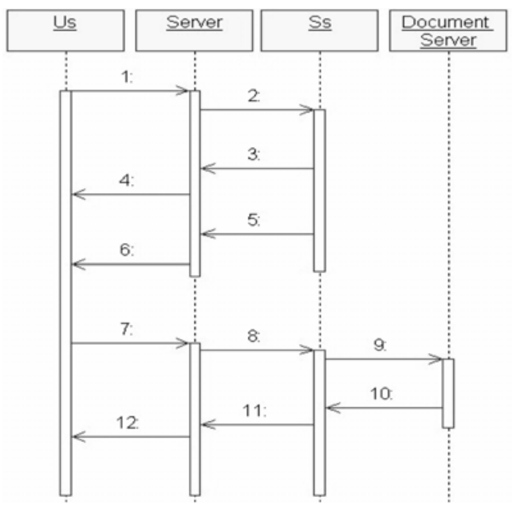
With the proxy-request mechanism described above, $S_s$ will receive all messages sent by $U_s$ and thus be aware of its dialogue status with $U_s$. Therefore, $S_s$ can detect and recover from potential errors. Since the participants of a dialogue are all aware of the dialogue status, the realisation of many advanced dialogue management mechanisms such as dialogue initiation and locking as described below then become possible.

### *3.2 Dialogue initiation and locking*

Another characteristic of a dialogue in a MUVE is that a user can only dedicate to a dialogue session at one time. From our daily experience, we know that the output voice from one-to-many people is common but input voice from many people to a person is unusual. For instance, when a teacher is giving a lecture to her students, the voice is one-to-many. When many students speak out for questions at the same time, it is difficult for the teacher to understand all the questions. Therefore, we think that for a valid dialogue, the output from an avatar to others may have a one-to-one or one-to-many relationship; but input from others to the avatar should only allow one-to-one relationship. To realise such a mechanism, we need to design a dialogue initiation and locking process to maintain dialogue states appropriately.

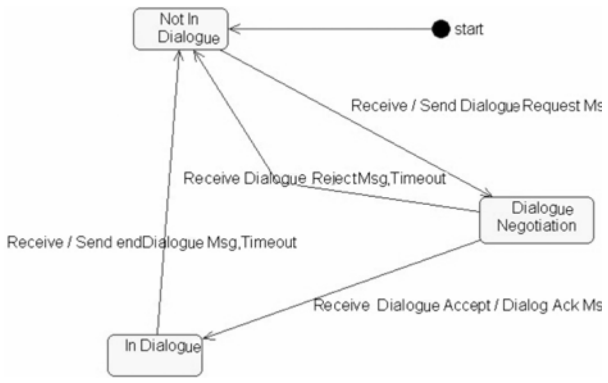
Before any clients can start their dialogues, they must negotiate with the other dialogue partner to ensure that it is not in a dialogue already. We have designed a negotiation process as illustrated in Figure 3. This process is semantically identical to the session establishment process described in the SIP specification (http://www.ietf.org/html.charters/sip-charter.html/). Despite the similarity, since the protocol representation of our design is XML-based and SIP adopts an ABNF-based syntax format, the protocol syntax and implementation of our system is different from the SIP standard.

**Figure 3** The sequence dialogue for the dialogue initiation process



To illustrate the dialogue session establishment process, assume that $U_s$ intends to have a dialogue with $S_s$. Firstly, $U_s$ has to confirm that it is not in a dialogue already with other clients in order to start the initiation process. $U_s$ will send an 'invite' message to $S_s$, and $S_s$ immediately responds with a 'negotiating' message (Step 1 in Figure 3). Meanwhile, $U_s$ will enter the 'dialogue negotiation' state (see Figure 4). If $S_s$ is also not in a dialogue, it will enter 'dialogue negotiation' state as well and return a 'dialogue accept' message (steps 5–6 in Figure 3) back to $U_s$. When $U_s$ receives this message, it will enter the 'in dialogue' state and send back a 'dialogue accept acknowledgement' message (steps 7–8 in Figure 3). $S_s$ then will also enter the 'in dialogue' state and fetch the first dialogue script from the document server for $U_s$ (steps 9–12 in Figure 3). On the other hand, if $S_s$ is busy in another dialogue already in step 4, it will send back a 'dialogue reject' message in step 5. When $U_s$ receives this message or the process times out due to any abnormal network problems, it will enter the 'not in dialogue' state and abort the initiation process.

**Figure 4** State diagram for dialogue initiation and locking


Not In Dialogue
start
Receive / Send DialogueRequest Ms
Receive Dialogue RejectMsg,Timeout
Dialogue Negotiation
Receive / Send endDialogue Msg,Timeout
Receive Dialogue Accept / Dialog Ack Ms
In Dialogue

### 3.3 *Dialogue message types*

The dialogue initiation messages described above are sent between the two engaging parties only. However, after the dialogue session starts, different avatars in a MUVE should receive different messages due to their distinguished roles in the dialogue. For example, except for the engaging avatars, the other avatars are observers of the dialogue. They should receive the content of the dialogue but should not participate or reply to any of these dialogues. Therefore, two types of messages are designed: *dialogue scripts* and *broadcasting scripts*. The dialogue scripts are similar to a typical VoiceXML dialogue form while the broadcasting scripts are like a dialogue without questions. The dialogue scripts are mandate and cannot be ignored while the broadcasting scripts may be safely ignored by other avatars if necessary according to the application. For example, if two pairs of avatars converse in the virtual environment at the same time, every avatars will receive the dialogue scripts from its dialogue partner and the broadcasting scripts from another pair. To be concentrated on their communication, they can choose to ignore the incoming broadcasting scripts. Although other types of messages can be extended, the functions of dialogue interrupts and three-way dialogues are not implemented in the current system design.

## 4 Design of XAML-V

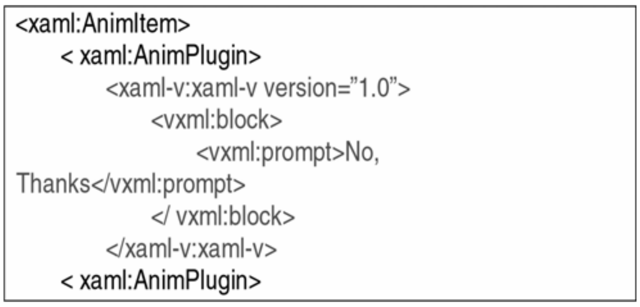
A scripting language called XAML-V, an extension of XAML, is designed to realise the VUI and dialogue management mechanism described in the previous section. In this section, we will present the scripting language in more details to illustrate how it takes advantage of the extensibility of XAML to make the animation scripting language speech-enabled. XAML-V mainly consists of tags with two types of functions: *dialogue context* and *dialogue management protocol*.

### 4.1 *Dialogue context*

XAML is an animation scripting language that allows other modules, such as XAML-V, to be incorporated as plug-ins. As shown in Figure 5, a XAML-V script is enclosed in the <xaml-v> tag, which is embedded in an

<AnimPlugin> tag. The dialogue context part of XAML-V is based on a subset of VoiceXML with the telephony-related elements removed since they are not appropriate in MUVE. For example, the tags of <block>, <prompt>, <form> and <field> all bear the same meanings as they are in VoiceXML while <form>, <submit> and <field> are redefined in XAML-V to achieve the new dialogue management mechanisms. The telephony-related tags such as the <transfer>, <filled> and <assign> tags are removed.

**Figure 5** XAML-V script as a plug-in of XAML

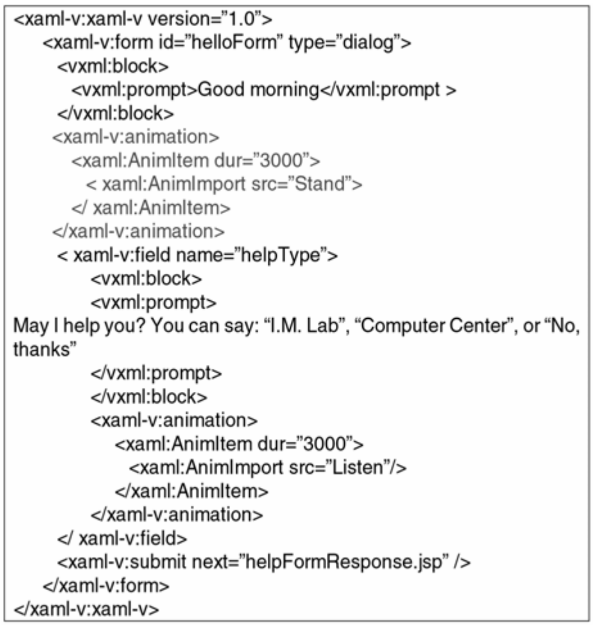


```
<xaml:AnimItem>
    < xaml:AnimPlugin>
        <xaml-v:xaml-v version="1.0">
            <vxml:block>
                <vxml:prompt>No,
Thanks</vxml:prompt>
            </ vxml:block>
        </xaml-v:xaml-v>
    < xaml:AnimPlugin>
```

In addition to the VoiceXML-related tags, XAML-V also supports embedded animations inside a dialogue at both the form level and the field level. The embedded animations are XAML scripts that do not recursively include XAML-V scripts. For example, in Figure 6, a form-level and a field-level animation that imports canned motions from external files through the <AnimImport> tag is used.

The XAML-V script example in Figure 6 describes a scenario where a computer-controlled avatar welcomes the user by a greeting statement ‘Good Morning, sir. May I help you?’ Then the system asks the user where he/she is interested in going while playing a high-level ‘listen’ animation clip at the same time to prompt the user for a response. The response will then be sent to the corresponding URL for further processing.

**Figure 6** Embedding animation in a XAML-V script

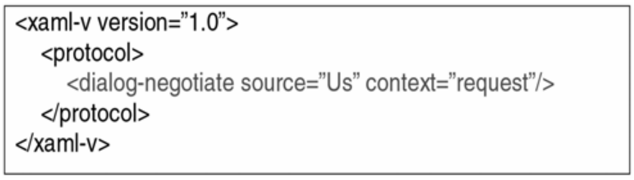


```
<xaml-v:xaml-v version="1.0">
   <xaml-v:form id="helloForm" type="dialog">
     <vxml:block>
       <vxml:prompt>Good morning</vxml:prompt >
     </vxml:block>
    <xaml-v:animation>
       <xaml:AnimItem dur="3000">
         < xaml:AnimImport src="Stand">
       </ xaml:AnimItem>
    </xaml-v:animation>
     < xaml-v:field name="helpType">
          <vxml:block>
          <vxml:prompt>
May I help you? You can say: "I.M. Lab", "Computer Center", or "No,
thanks"
          </vxml:prompt>
          </vxml:block>
          <xaml-v:animation>
              <xaml:AnimItem dur="3000">
                <xaml:AnimImport src="Listen"/>
              </xaml:AnimItem>
          </xaml-v:animation>
     </ xaml-v:field>
     <xaml-v:submit next="helpFormResponse.jsp" />
   </xaml-v:form>
</xaml-v:xaml-v>
```

## 4.2 *Dialogue management protocol*

Several tags are added to support the dialogue management mechanism proposed in the previous section. Figure 7 shows an example of dialogue negotiation message. The ‘context’ attribute indicates the type of dialogue negotiation being executed, and the ‘source’ attribute indicates where this message is from.

**Figure 7** Dialogue request message

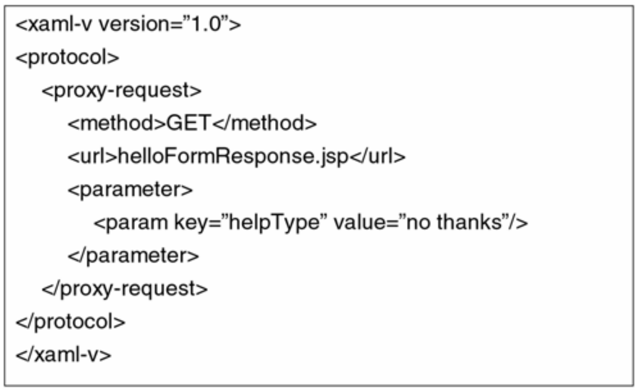


```
<xaml-v version="1.0">
   <protocol>
      <dialog-negotiate source="Us" context="request"/>
   </protocol>
</xaml-v>
```

In Figure 7, the ‘context’ attribute is ‘request’ and the ‘source’ is ‘$U_s$’. The script means that an avatar ‘$U_s$’ would like to ‘request’ a conversion to the user. The possible values for the ‘context’ attribute of the dialogue-negotiation element include: *request*, *accept*, *reject*, *dialogAck* and *endDialog*. Each of these values maps to an action in a dialogue negotiation process described in the previous section.

Figure 8 shows an example of the proxy-request mechanism in XAML-V. The idea is to encapsulate HTTP GET/POST messages in the <proxy-request> tag such that the system-controlled avatar can fetch the next document from the document server. In the <proxy-request> element, the HTTP method, requesting URL and requesting parameters are the sub elements to encapsulate detail information.

**Figure 8** An XAML-V script for proxy request



```
<xaml-v version="1.0">
<protocol>
   <proxy-request>
      <method>GET</method>
      <url>helloFormResponse.jsp</url>
      <parameter>
         <param key="helpType" value="no thanks"/>
      </parameter>
   </proxy-request>
</protocol>
</xaml-v>
```

## 5 Implementation and example

### 5.1 *Implementation*

We have implemented the enhanced dialogue model with XAML-V in IMNet (Li et al., 2004, 2005). The XAML-V module serves as a plug-in component of the XAML platform and coordinates with various input and output devices. The XAML-V module interprets XAML-V script and manages several dialogue mechanisms (e.g. dialogue lock or dialogue state). A comparison of the implementation of XAML with other speech-enabled

MUVE systems is summarised in Table 1. The main differences are on the adopted script-based language (XAML-V) for dialogue flow control and how it is realised in the 3D-virtual environment.

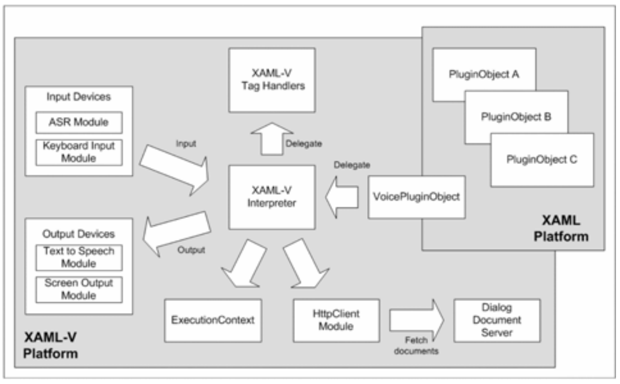
**Table 1** Comparison of implementation in various MUVE's with VUI

| *System* | *Cernak and Sannier (2002)* | *Wauchope MSFT (2003)* | *Wauchope et al. ISFS (2003)* | *XAML-V* |
|---|---|---|---|---|
| Virtual environment | VRAC's C6 | EA's World Toolkit | Cortona VRML Browser | IM-Browser (Li et al., 2005) |
| Speech recognition | CSLU Toolkit | IBM ViaVoice 8 | IBM ViaVoice 8 | IBM ViaVoice 9 |
| SR grammar | Home made | IBM SRCL | JSGF | SRGF (W3C Standard) |
| SR invocation | Keyword | Not mentioned | Push to talk | Push to talk |
| TTS | Festival | IBM ViaVoice 8 | IBM ViaVoice 8 | IBM ViaVoice 9 |
| Speech API | Not mentioned | IBM SMAPI | JSAPI | JSAPI w/ Cloud Garden Bridge |
| Speech-VR bridge | TCP Socket | TCP Socket | UCP Socket | TCP Socket |
| Dialogue flow control | SCI IDE | Rule and data stored in RDBMS | Rule and data stored in RDBMS | XAML-V |

Figure 9 shows the overall architecture of XAML-V platform. The *VoicePluginObject* serves as a plug-in point to XAML platform. It accepts scripts from the XAML platform and delegate to a XAML-V interpreter. The *XAML-V interpreter* is the core of the XAML-V platform, which parses incoming scripts and orchestrates the other components. *ExecutionContext* is the data store for run-time configurations and information needed by the interpreter. The *dialog*ue *document server* is a repository for dialogue scripts. These scripts may also be generated dynamically using server-side scripting technologies. For example, we use an open source Java Servlet container (Tomcat 4.1) as the dialogue document server in our implementation. The *HttpClient* fetches dialogue scripts from the document server and handle HTTP protocol details for the interpreter. *Tag Handlers* are collections of classes conformed to a 'TagHandler' interface and each of them is designed to handle a specific tag. The interpreter delegates work to this component according to the tags that it encounters. For example, it will delegate work to PromptTagHandler class if the interpreter encounters a <prompt> tag. In addition to rendering the voice with the TTS module, the PromptTagHandler object will send out a broadcasting message containing a <prompt> script to let all other avatars render the voice as observers.

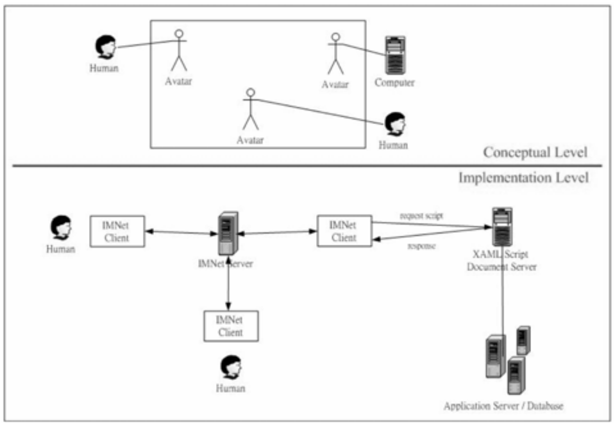
According to the plug-in model of XAML, when the interpreter encounters the <AnimPlugin> element, it will search a preconfigured component registry for a valid plug-in to handle the script described inside the <AnimPlugin> element. The XAML interpreter will acquire the control of current executing thread and delegate to a plug-in component when it finds one. Since XAML-V is a plug-in component, the XAML-V interpreter will take over the control of current thread and continue to execute the script.

**Figure 9** System architecture of XAML-V platform



To evaluate the effectiveness of the proposed mechanism, we have built a prototype system for experiments. The experiments consist of three client avatars, two controlled by the human and one controlled by the computer. The layout of the experimental environment is shown in Figure 10. The IMNet clients exchange their messages via the IMNet server implemented based on the OpenJMS[2] running on a computer with Intel Pentium 4 1.6 GHz CPU and 768 MB of memory. In order to generate dialogue scripts according to the user's response dynamically, we have constructed the Document Server with the Apache HTTP server and the Tomcat application server, which act as the server-side script generator.

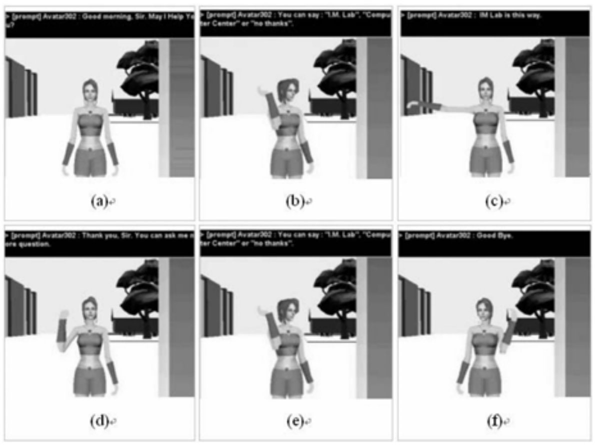
**Figure 10** Experimental environment



## 5.2 An example

In Figure 11, we show the snapshots of the user interface for an example of interactive animation with a voice dialogue written in XAML and XAML-V. The example dialogue script is similar to the one shown in Figure 6. In the scenario, a virtual character acts as a receptionist via the VUI when a real user enters the virtual environment. The client application will start dialogue negotiation by issuing a dialogue request to the receptionist's client and entering the 'dialogue negotiation' state (see Figure 4 and steps 1–4 in Figure 3). In this scenario, we assume the receptionist is not having a dialogue with another avatar, so the user's client will receive a 'dialogue accept' message and enter the 'in dialogue' state (see Figure 4 and steps 5–6 in Figure 3). After the script is fetched and sent to user's client (steps 9–12 in Figure 3), the XAML-V interpreter will start playing this script on the user's client machine.

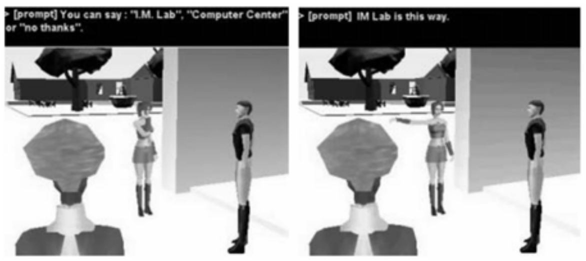
**Figure 11** Snapshots of the interface for an example dialogue in a MUVE


> [prompt] Avatar302 : Good morning, Sir, May I Help You?
> [prompt] Avatar302 : You can say : "I.M. Lab", "Computer Center" or "no thanks".
> [prompt] Avatar302 : IM Lab is this way.
> [prompt] Avatar302 : Thank you, Sir. You can ask me more question.
> [prompt] Avatar302 : You can say : "I.M. Lab", "Computer Center" or "no thanks".
> [prompt] Avatar302 : Good Bye.
(a)
(b)
(c)
(d)
(e)
(f)

The receptionist greets the guest by saying "Good morning, Sir, May I help you?" (Figure 11(a)) Then she will listen to the user's input for the destination that he/she is interested in and play a high-level animation "listening" at the same time (Figure 11(b)). If the user does not need any assistance, the receptionist will end the dialogue by saying "Good-Bye" (Figure 11(f)). If the user specifies one of the destinations that the receptionist knows, she will guide the user to the destination (Figure 11(c)). Unless the user says "No, thanks", the receptionist will continue to ask the user for further question (Figures 11(d) and (e)). Note that the receptionist's reaction to the user's utterance is decided by posting (HTTP POST) the speech recognition results back to the document server. In Figure 6, the "next" attribute of <xaml-v:submit> tag is used for this purpose.

Figure 12 shows two snapshots (corresponding to (b) and (c) in Figure 11) of the dialogue from the observer's view. The avatar with blonde hair is the observer of this dialogue. She can hear all speech voices of the dialogue, or may choose to ignore these voices safely if she would like to have a dialogue with another avatar or if she is too far away from them according to the specific definition of the application.

**Figure 12** Snapshots (b and c) of the above dialogue from an observer's viewpoint


> [prompt] You can say : "I.M. Lab", "Computer Center" or "no thanks".
> [prompt] IM Lab is this way.

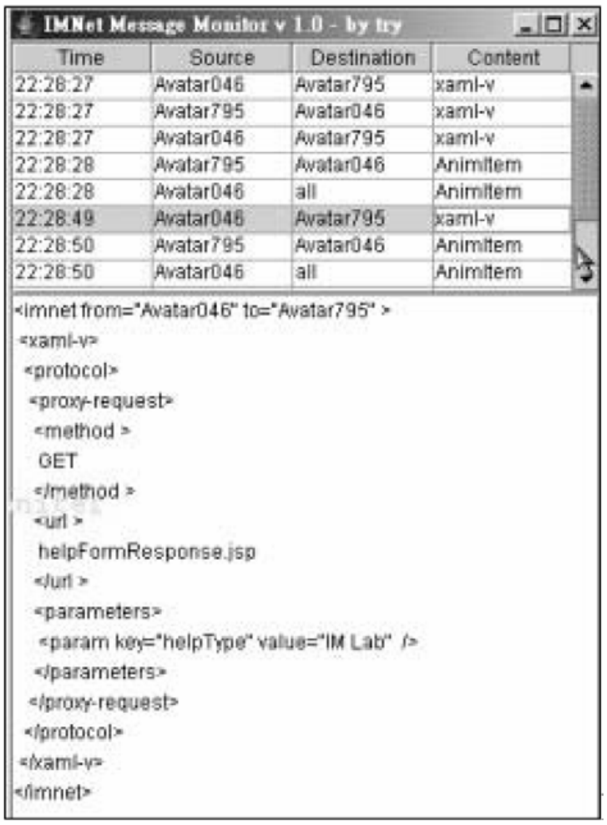
## 5.3 Experiments and evaluation

We evaluate the functional correctness of the implemented system by testing the pre-defined scenario in Section 5.2 with the following three different use cases:[3]

- *Use case 1*: human–computer interaction with text input and speech output interface. A user communicates by typing with the computer that responds with voice dialogues.
- *Use case 2*: human–computer interaction with speech input and speech output interface. A user communicates directly with the computer by speaking to the computer that understands a limited amount of vocabulary.
- *Use case 3*: human–human interaction with text input and speech output interface. A user communicates with another user by typing while the responses are all through voice interfaces.

We have also developed a Message Monitor, as shown in Figure 13, to intercept and log all messages passing through the IMNet server and to ensure that each interaction among clients generates correct XAML-V messages. Our experiments show that the IMNet server can process 1200 messages per second on the current experimental platform. If the server is overloaded, message delivery will be ignored until the traffic congestion is relieved. Although the current implementation only allows two parties to speak to each other in a dialogue, there could be many other clients listening to the dialogue. The maximal number of clients supported will depend on number of concurrent dialogues and the number of participants (active parties or audience) in each dialogue.

**Figure 13** The message monitor



| Time | Source | Destination | Content |
|---|---|---|---|
| 22:28:27 | Avatar046 | Avatar795 | xaml-v |
| 22:28:27 | Avatar795 | Avatar046 | xaml-v |
| 22:28:27 | Avatar046 | Avatar795 | xaml-v |
| 22:28:28 | Avatar795 | Avatar046 | AnimItem |
| 22:28:28 | Avatar046 | all | AnimItem |
| 22:28:49 | Avatar046 | Avatar795 | xaml-v |
| 22:28:50 | Avatar795 | Avatar046 | AnimItem |
| 22:28:50 | Avatar046 | all | AnimItem |

```
<imnet from="Avatar046" to="Avatar795" >
 <xaml-v>
  <protocol>
   <proxy-request>
    <method >
    GET
    </method >
    <url >
    helpFormResponse.jsp
    </url >
    <parameters>
     <param key="helpType" value="IM Lab" />
    </parameters>
   </proxy-request>
  </protocol>
 </xaml-v>
</imnet>
```

## 6 Conclusions and future work

In this paper, we have proposed to enhance MUVE with a VUI. We have presented a dialogue management mechanism for MUVE based on VoiceXML and XAML. The proposed XAML-V dialogue scripting language includes functions on dialogue lock, dialogue broadcasting, dialogue negotiation, and a proxy-request mechanism. We have demonstrated the appropriateness of this design by examples and shown that by integrating with an appropriate voice interface, users can communicate with each other in a more natural way in MUVE.

We have been focusing on realising the dialogue management mechanism for MUVE; however, many desirable features still need to be added to enhance the immersion of the virtual environment. For example, the volume of the voice dialogue as well as other 3D sound effects should be adjustable according to the relative spatial locations between avatars. In addition, a more attractive facial animation synchronised with the voice dialogue should be adopted to enhance visual realism. We also would like to extend the dialogue model to allow interrupts and dialogues with several participants at the same time.

## References

Apaydin, O. (2002) 'Networked humanoid animation driven by human voice using extensible 3D (X3D), H-Anim and Java speech open standards', Master's Thesis, Naval Postgraduate School.

Cernak, M. and Sannier, A. (2002) 'Command speech interface to virtual reality applications', *Technical Report*, Virtual Reality Applications Center at Iowa State University of Science and Technology.

Descamps, S., Prendinger, H. and Ishizuka, M. (2001) 'A multimodal presentation mark-up language for enhanced affective presentation', *Proceedings of the International Conference on Intelligent Multimedia and Distant Education (ICIMADE-01)*, Advances in Educational Technologies: Multimedia, WWW and Distance Education, pp.9–16.

Frecon, E. and Stenius, M. (1998) 'DIVE: a scalable network architecture for distributed virtual environments', *Distributed Systems Engineering Journal* (special issue on *Distributed Virtual Environments*), Vol. 5, No. 3, pp.91–100.

Greenhalgh, C. and Benford, S. (1995) 'MASSIVE: a collaborative virtual environment for teleconferencing', *ACM Transactions on CHI*, Vol. 2, pp.239–261.

Huang, Z., Eliens, A. and Visser, C. (2002) 'STEP: a scripting language for embodied agents', *Proceedings of the Workshop on Lifelike Animated Agents*.

Kshirsagar, S., Guye-Vuilleme, A. and Kamyab, K. (2002) 'Avatar markup language', *Proceedings of Eighth Eurographics Workshop on Virtual Environments*, pp.169–177.

Li, T.Y., Liao, M.Y. and Liao, J.F. (2004) 'An extensible scripting language for interactive animation in a speech-enabled virtual environment', *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME2004)*, Taipei, Taiwan.

Li, T.Y., Liao, M.Y. and Tao, P.C. (2005) 'IMNET: an experimental testbed for extensible multi-user virtual environment systems', in O. Gervasi et al. (Eds). *ICCSA 2005, LNCS 3480*, Springer-Verlag Berlin Heidelberg, pp.957–966.

Matijasevic, M. (1997) 'A review of networked multi-user virtual environment', Available at: http://citeseer.nj.nec.com/matijasevic97review.html.

Nyberg, E., Mitamura, T., Placeway, P., Duggan, M. and Hataoka, N. (2002) 'DialogXML: extending VoiceXML for dynamic dialog management', *Proceedings of the Human Language Technology Conference*.

Sagayama, S., et al. (2003) 'Galatea: an anthropomorphic spoken dialogue agent toolkit', *IPSJ SIG-SLP*.

Wauchope, K. (2003) 'Interactive ship familiarization system: technical description', *AIC Technical Report AIC-03-001*, Navy Center for Applied Research in Artificial Intelligence, Washington, DC.

Wauchope, K., Everett, S., Tate, D. and Maney, T. (2003) 'Speech-interactive virtual environments for ship familiarization', *Proceedings of Second International Euro Conference on Computer and IT Applications in the Maritime Industries (COMPIT '03)*, Hamburg, Germany, pp.70–83.

## Notes

[1]Available at: http://www.w3.org/TR/xslt.

[2]Available at: http://openjms.sourceforge.net/.

[3]The video clips for the experiments of these use cases can be downloaded online at: http://www.try.idv.tw/research/.