Dialogue Session Management Using VoiceXML

Augustine Tsai, Andrew N. Pargellis, Chin-Hui Lee, Joseph P. Olive

Multimedia Communication Laboratory
Bell Labs, Lucent Technologies
Murray Hill, NJ 07974, USA
{atsai,anp,chl,jpolive}@research.bell-labs.com

Abstract

A spoken dialogue system capable of maintaining a human-machine conversation over a telephone must simultaneously maintain many dialogue sessions with different callers and third-party servers running applications over the Internet. There are three main issues to be addressed. First, the caller's identity has to be passed between each dialogue turn. Secondly, the dialogue state and connections to third party servers have to be continuously maintained. Thirdly, the system interfaces have to follow some standards, especially Internet protocols. We use session tickets and session object hashes to address these issues. Each dialogue turn is encoded in a session data object and VXML page. The session tickets contain the session ID and security-related information passed between the VoiceXML interpreter and document server. The session object hash stores the third party connection handle, which is retrieved for subsequent dialogue turns.

1. Introduction

The design of spoken dialogue systems capable of maintaining human-machine conversations by telephone is a challenging one. This is especially true when there are many simultaneous callers, and the system must maintain contact with third-party application servers spread out over a distributed network using Internet protocols. Such a dialogue system needs to maintain caller identity and maintain continuous connections to external servers using standardized protocols.

We model a conversation, consisting of a series of dialogue turns, as a collection of dialogue states with allowed transitions between them. Previously, we used a Voice User Interface (VUI) platform to dynamically specify each dialogue turn by abstracting dialogue states for different tasks and services [1]. Specifically, in this paradigm each subtask within a domain consists of a set of specifications such as a system prompt, grammar, and control logic for performing the requested action. A dialogue manager uses a user's profile and dialogue session history, to respond to a user's requests in a dynamic and personal manner [2]. The VUI platform allows the integration of information retrieval, telephony, transaction, and messaging services [1,2], while providing persistent connections between speech engines and third party servers. The speech interface is the Bell Labs Speech Technology Integration Platform (BL-STIP), with proprietary protocols and APIs [3]. The VUI uses a set of Voice Interface Language (VIL) commands to interface with BL-STIP [4]. A dialogue session persists, while interfacing with third-party servers, because all interactions are in the form of function calls to routines within the VUI platform. The data exchanged are conducted through function calls or socket connections. The speech recognition modules and dialogue manager work together as one software entity, and this entity implicitly maintains a continuous dialogue session.

It is desirable to use a standardized format for specifying dialogue sessions held with agents distributed over a network such as the Internet. An international standard, the Voice eXtensible Markup Language (VoiceXML or simply, VXML) has been developed recently [5] and is now receiving broader industrial acceptance [6]. It gives the interactive voice presentation, while the Hyper Text Markup Language (HTML) gives the text/graphics presentation. There are many commercial VXML development tools in the market including BeBocal Café, WebSphere Voice Server SDK, Mobile ADK, Nuance V-Builder, Tellme Studio, and VoiceGenie. Most of these tools provide only static VXML file loading, validating, and visual editing capability.

Therefore, we present a VXML-based dialogue session management paradigm for providing a voice portal to integrated messaging and information retrieval services. In the following sections we discuss how to maintain a VXML dialogue session using a session ticket and associated session object hash. Finally, we discuss the dynamic generation of VXML pages using a stylesheet specification (XSLT).

2. Session Management

VXML enables integration of the voice services with data services using the familiar client-server paradigm. A dialogue session is composed of a sequence of interactive dialogue turns. We encode each dialogue turn as a session data object and VXML page. The VXML page is dynamically written by a dialogue manager on an external web server. The web server maintains the overall service logic, performs database and legacy system operations, and produces a dynamic dialogue. A VXML page specifies each interaction dialogue to be conducted by a VXML interpreter. User input is processed by the dialogue manager and submitted to the web server. The web server may reply with another VXML page to continue the user's dialogue session.

Figure 1 shows the interaction between three types of servers. A user's call, originating at the lower left, is picked up by a telephony server on any speech platform containing a VXML interpreter and speech engines. We use the *Lucent Speech Server* (LSS) platform [7], shown by the large block on the left. The LSS platform consists of a multi-threaded telephony server, an automatic speech recognizer (ASR), text to speech (TTS) synthesizer, and a VXML interpreter. The call initiates a session, controlled by the *Session Manager* (SM), shown by the *web-server* process in the right-hand block. During a dialogue session, the SM will interact with a Dialogue Manager (DM) that is responsible for processing a caller's spoken requests. The SM may also interact with an

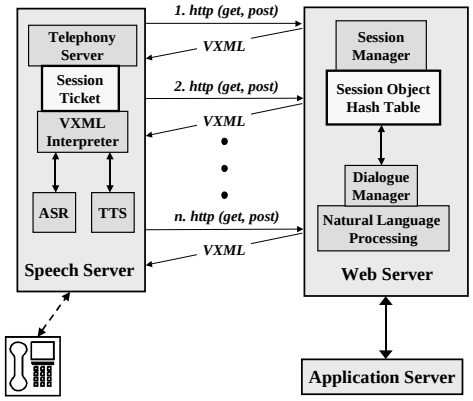

Figure 1. Session ticket and session object hash.

application server, shown schematically by the lower right block. These interactions are over the Internet, using whatever protocols are required by the remote servers. For example, information retrieval from www sites will use an http protocol, whereas a messaging service to Lucent's Audix server would use a pop3 protocol.

The web server (VXML document server) responds to each client (VXML Interpreter) request without relating that request to previous or subsequent requests. The request-response-disconnect nature of HTTP precludes any sharing information between the client and the server. The SM must allow clients and servers to exchange information by placing HTTP requests and responses within the larger context of a session.

There are two issues to be addressed in order to accomplish this objective. First, the caller's identity has to be passed between each dialogue turn. The caller's identity has to be shared by multiple VXML pages, one for each individual dialogue turn, since the actual spoken portion of each dialogue turn is encoded in a separately generated VXML page. Secondly, the dialogue state and any connections to third party application servers have to be continuously maintained during the dialogue session. We don't want to re-login to the application servers for each dialogue turn.

The VXML specification has addressed the first issue to some extent. Multiple VXML pages can share a root page using the *application* tag (e.g. vxml version ="1.0" application = "root.vxml"). Therefore, the caller's identity can be defined in the root page and shared by other VXML pages. The solution is suitable for the static pages but not for the dynamically generated pages.

2.1. Session Ticket and Session Object Hash

We use session ticketes and a session object hash as solutions to the above-mentioned issues. The session tickets are sent by the session manager and are validated when the client submits back to the session manager. Session tickets contain the information of the caller id, client IP address, timeout flag, and security information. Each session ticket contains attribute-value pairs. For example, the attribute could be the session id and its value would be the id assigned to the current session with the caller. The session tickets are stored in the VXML interpreter as shown in the Figure 1. The session ticket is a set of persistent cookies [8]. In order to maintain the persistent connection to the application server over multiple dialogue turns (each turn contains a distinct HTTP request-response transaction), we employ the session object hash to store the application object handle. The session object hash may also store a history stack of dialogue states. The session object hash is stored in the SM on the web server.

An example of part of a session-object hash is shown below in Table 1 for the case where a caller is listening to an audio rendering of their email messages. This hash was generated, with help from the web-server's DM, after the caller made the request,

I want to forward the current message.

The *domain* is the particular service the caller is presently using and the *subtask* is a specific task within the current domain. In order to carry out a subtask such as "forward the message", several attributes may be required. In this example, both a *recipient* and message number (*msgNum*) are required. The recipient's name was not given, so the DM's control logic for the current dialogue state must identify this omission and modify the outgoing system prompt (*sysPrompt*) accordingly [1,2].

Table 1: A sample session-object hash.

Key	Value
domain	email
subtask	forwardMsg
recipient	[null]
msgNum	[current]
grammarSrc	recipient.gram
sysPrompt	What is the recipient's name?

2.2. An Example for Retrieving Email

The email retrieval example illustrates the use of a session ticket and session object hash. The *server* is the Session Manager and the *client* is the VXML interpreter. Each client request and SM response constitutes an HTTP transaction, which is part of the dialogue turn in a dialogue session. The first six http transactions are outlined below:

1. Client : *Get http://localhost/servlet HTTP/1.0*

2. Server: *HTTP/1.1 200 OK*
   Set session ticket, name=session_id, value="xxx"
   Response-> login.vxml page.

3. Client: *Post /localhost/servlet HTTP /1.0*
   parameter="usernam" value="john"
   parameter="password" value = "xxxx"

4. Server: *HTTP/1.0 200 OK*
   Connect pop3 email server.
   Session.storeObjectHash(pop3_handle)
   Response->menu.vxml

5. Client: *Post /localhost/servlet HTTP /1.0*
   parameter = "action" value = "play email header"

6. Server: HTTP/1.1 200 OK
   session.getObjectHash(pop3_handle)
   Respond-> mail_content.vxml

In ***Step 1***, the user places a phone call to the speech server, the URL associated to this phone number is requested. In ***Step 2***, the session server receives the incoming http request. The initial VXML page, *login.vxml*, and session ticket are created and sent back to the client. This session ticket is a session ID that is used to identify the caller in the next http request. In ***Step 3***, the client receives the vxml page that is parsed by the interpreter. The user is prompted for the input information, the user utterance is processed, and the vxml form is filled and submitted using http POST. In ***Step 4***, the SM parses the submitted parameter string for the login information. Next, the SM logs onto the pop3 application server and obtains the object handle. The object handle is stored in the session object hash. The *menu.vxml* page is created and in ***Step 5***, the caller selects a menu action and the client submits it. In ***Step 6***, the SM retrieves the application object handle from the session object hash. The object handle is used in future dialogue turns to retrieve individual email messages.

3. Maintaining a Single Dialogue Turn

During a dialogue session, both participants must have an opportunity to respond to, or make a request of, the other party during each of many dialogue turns. Each turn is initialized by accessing a representation of the current dialogue session, as well as a representation of the overall conversation, which includes a history of all previous turns. Since many callers may be interacting with the computer agent simultaneously, the SM must provide the session ID, a critical component in maintaining the integrity and continuity of a particular conversation with a caller.

A single dialogue turn in this conversation follows the flow shown in Fig. 2, starting with the upper left hand block labeled *Session ID*. The session ID, unique to each individual phone call, is used by the Dialogue Manager module (shown above in Fig. 1) to identify the current dialogue state specifications, stored in a session object hash, as well as download the most recent input from the correct user.

In order to process the input string from the user, the DM must create a series of grammar rules and specifications that are read in from several sources. The DM uses the session ID to determine the current subtask being performed. In the email example described above in Table 1, the user's input was a request to forward the current message. This utterance must be processed according to some grammar rules. These rules can be read from some specification files. Therefore, the DM downloads all specification files for the email subtask. This step is shown in Fig. 2 by the *Read Specifications* block, with the set of specification files (which may be static files from a library or dynamically generated by an external server) shown schematically by the file folders. The user's input is then processed by the *Natural Language Processor* (NLP) module. In the simplest case, the NLP unit could merely consist of concept tags with corresponding phrases. The input utterance

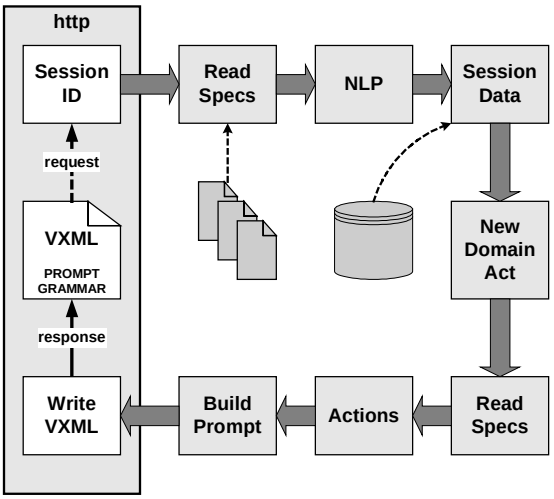

Figure 2. Flow of the dialogue manager for a single dialogue turn, starting with getting the Session ID, upper left block.

is then parsed, replacing all such phrases with a key::value pair in a hash table, with the key being the concept and the value the phrase. The user's input must be processed for its overall intent and any attribute values associated with that intent. The user's intent is the next subtask to be performed by the system. The attributes, and their values, are used to accomplish that subtask. For the email domain, the next subtask may be "forwardMessage" and the required attributes are "messageNumber" and "Recipient". The SM then updates the session database with the new information. This includes the next subtask (forward the message) to be performed by the system, shown by the *New Domain Act* block. The next subtask may be unchanged, or it may be a new subtask in a new service (weather forecasts or banking transactions).

The specification files for the next dialogue turn must be accessed since the system will have to build an outgoing prompt to the user and identify any new parameters for the speech engines, such as a new grammar. This is shown by the second *Read Specifications* block, in the lower right of Fig. 2. The *Actions* block performs any required actions. For example, if the user had asked for a weather forecast, the DM would have to access an external database, probably over the Internet, from which to retrieve, and then process, the required information. The DM then creates an outgoing text string, to be sent to the speech engine, shown schematically by the *Build Prompt* block. Finally, the DM writes a dynamic vxml page that is then sent to the VXML interpreter on the speech platform.

The dialogue turn ends with a new user utterance that the vxml interpreter submits as an input to the Session Manager to be used in the next dialogue turn. In this way, all language processing of the user's input can be handled by natural language processing modules of arbitrary complexity, while the actual spoken interface is handled by the dynamically generated vxml pages.

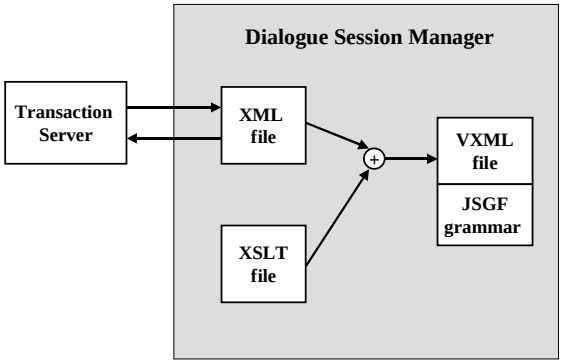

Figure 3: Use XSLT to generate dynamic VoiceXml

4. Dynamic VXML generation using XSLT

It is important that the dialogue Session Manager has a universal XML interface for data and information retrieval since an industry trend is to store more data content in the XML format. We need a mechanism to interpret the XML document and convert it to the VXML document. The eXtensible Stylesheet Language Transformation (XSLT) provides a good solution to accomplish this goal. The XSLT can convert an XML document to any of the markup languages such as VXML, HTML, and WML. The XML document includes a header as shown in the following *message.xml* example. The default style sheet is for HTML file generation. However, if the accessing device is a VXML interpreter, then the VXML style sheet will be used. Figure 3 shows that the transaction server provides a XML file. The SM then uses XSLT to convert this file into a VXML document.

Here is an example showing how the XSLT extracts the senders from the *message.xml* document. The for-each construct in the *vxml.xsl* file matches the "message" tag and extracts the sender field from each message. *Sender.xml* is the resultant output file.

message.xml

```
<?xml version="1.0">
<?xml-stylessheet href="html.xsl" type="text/xsl"?>
<?xml-stylessheet href="vxml.xsl" … media="vxml"?>
<folder>
<message>
    <type>email</type>
    <sender>john smith</sender>
    <subject>meeting</subject>
    <content>The group meeting will be held…….</content>
</message>
<message>
   <type>audix</type>
   <sender>Peter Goodman </sender>
</message>
```

vxml.xsl

```
<?xml verstion="1.0"?>
<xml:stylesheet
xmlns:xsl=//www.w3.org/1999/XSL/Transform>
 <xsl:template match = "folder">
 <?xml version="1.0">
    <vxml version="1.0">
    <block>you have email from</block>
     <xsl: for-each select="message">
      <prompt>
         <xsl:value-of select="from" />
      </prompt>
     </xsl:for-each>
</vxml>
```

sender.vxml

```
<?xml version="1.0">
<vxml version="1.0">
    <block>You have email from</block>
    <prompt> John Smith </prompt>
    <prompt> Peter Goodman </prompt>
</vxml>
```

5. Conclusions

We have built a VXML-based prototype system that uses a dialogue Session Manager for providing integrated messaging and information retrieval services. A session ticket is stored at the client (speech-server) side and a session object hash is stored in the session manager on the server side. The SM maintains a dialogue session by means of a session ticket and session object hash. Individual dialogue states are specified dynamically and an XSLT module is being designed to enable the dialogue manager to exchange data in a universal XML format with other application servers.

6. References

[1] A. Pargellis, H.-K. J. Kuo, C.-H. Lee, "Automatic Application Generator Matches User Expectations to System Capabilities", *Interactive Dialogue in Multi-modal Systems*, Kloster Irsee, Germany; 22– 24 June 1999

[2] A. Pargellis, H.-K. J. Kuo, C.-H. Lee, "Automatic Dialogue Generator Creates User Defined Applications", *Proc. of the Sixth European Conf. on Speech Comm. and Tech.*, Budapest, 1999, 3:1175-1178

[3] Q. Zhou, A. Sadd, and S. Abdou, ``An Enhanced BLSTIP Dialogue Platform", *International Conference for Spoken Language Procession***,** Beijing, 2000, 3:1061-1064

[4] A. Pargellis, Q. Zhou, A. Saad, C.-H. Lee, "A Language for Creating Speech Application", *ICSLP'98*, Sydney, Australia; 01–04 December 1998

[5] VoiceXML Forum, *Voice eXtensible Markup Language VoiceXML*, URL: www.voicexml.org, 2000

[6] *World Wide Web Consortium*, URL:www.w3.org

[7] *Lucent Speech Server*, URL:www.lucyipa.com/document

[8] IETF Request for Comments 2109: HTTP State Management Mechanism, 1997