A Development Tool for VoiceXML-Based Interactive Voice Response Systems

Cheng-Hsiung Chen  Nai-Wei Lin
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan 621, R.O.C.
{ccs88m,naiwei}@cs.ccu.edu.tw

Abstract

This paper describes a development tool for VoiceXML-based interactive voice response systems. This tool provides a set of friendly graphical user interfaces to relieve users from writing code. This set of graphical user interfaces supports a set of commonly used dialog patterns to facilitate the design of the dialog structure of the system. This tool also provides a set of predefined asynchronous dialog transfer events to facilitate the transfer among dialogs.

1. Introduction

An interactive voice response (IVP) system allows a server to interactively respond to clients by voice via telephones. VoiceXML is a dialog markup language that is designed to facilitate the development of interactive voice response services using World Wide Web technology [4]. This paper describes a tool TIVR for developing VoiceXML-based interactive voice response systems.

This tool provides a set of friendly graphical user interfaces to relieve users from writing code. This set of graphical user interfaces supports a set of commonly used dialog patterns to facilitate the design of the dialog structure of the system. This tool also provides a set of predefined asynchronous dialog transfer events to facilitate the transfer among dialogs.

The remainder of this paper is organized as follows. Section 2 gives a brief introduction to VoiceXML-based IVP systems. Section 3 provides an overview of the TIVR development tool. Section 4 describes the implementation of the dialog design patterns. Section 5 describes the implementation of the asynchronous dialog transfer events. Finally, conclusions are given in Section 6.

2. VoiceXML-based IVP systems

The world wide web provides easy access for abundant visual information in the world. However, there are many more people who have access to telephones than people who have access to computers. Hence, it is also beneficial to provide easy access for abundant voice information in the world. In addition, for visually impaired people, it becomes very significant to provide easy access for abundant voice information in the world.

The IVP systems have been used to provide easy access to abundant voice information. The user can use a telephone to call a number to interact with a server via voice. A well-designed IVP system can usually provide good quality information access services for users.

Traditional IVP systems are implemented using general-purpose programming languages. This makes the development and maintenance of the IVP systems difficult. Recently, a domain-specific programming language VoiceXML has been proposed to facilitate the development and maintenance of IVP systems.

A typical architecture for VoiceXML-based IVP systems is given in Figure 1. The voice gateway is used to handle the telephone calls from the users. After receiving a call, the voice gateway will request the application root VoiceXML document from the web server and invoke the VoiceXML interpreter to interpret this document. The VoiceXML interpreter may use the Text-to-Speech (TTS) component to translate a segment of text into speech and sent it to the caller. The VoiceXML interpreter may also use the Automatic-Speech-Recognition (ASR) component to recognize the speech from the caller, access the web server or database server according to the recognized request, and then respond the result using the TTS component.
An IVP application consists of a set of VoiceXML documents. Each VoiceXML document is a vxml element that contains one or more dialogs. A dialog may present the user with speech information or prompt the user to provide speech or touch-tone information. A dialog is a form element or a menu element. A form is a dialog for presenting information and collecting data. A menu is a dialog for choosing amongst alternative destinations. A dialog may direct the flow of control to another dialog in the same document, to a dialog in another document in the same application, or to a dialog in another application.

A dialog may be structured as a sequence of subdialog to simplify the dialog and facilitate the reuse of subdialogs. The flow of transfer among dialogs and subdialogs is depicted in Figure 2. A dialog may transfer the flow of control to a subdialog using the subdialog element. After finishing its task, the subdialog may transfer the flow of control back the dialog using the return element.

3. An overview of TIVR

The tool TIVR is designed to ease the development of IVR systems. It provides a set of friendly graphical user interfaces to specify the structure of dialogs in an IVR system. This allows the users to develop IVR systems without writing code.

The graphical user interfaces can be divided into
three categories of windows: database connection window, application initiation window, dialog design windows, and system simulation window.

The database connection category is used to generate code to open and connect to a specific database. The tool uses the Microsoft universal data access standard Open database Connectivity (ODBC) to allow the generation of code independent of backend databases [1].

The application initiation category is used to initiate a specific application. All the VoiceXML documents within an application will be created and stored in the same directory of the file structure. A window is provided to allow the users to browse the file structure.

The dialog design category is used to design the dialog structure of the system. It contains six dialog patterns: key_input, voice_input, search, verification, update, and menu. These dialog patterns can generate either static VoiceXML documents or php codes that dynamically generate VoiceXML documents [2]. The implementation of these dialog patterns will be described in detailed in the next section.

The system simulation category is used to simulate the functionality of the developed IVP system using MyVoiceGenie [3].

4. Dialog design patterns

This section describes the implementation of the six dialog design patterns. The key_input dialog pattern uses the form element to input data from touch-tone keys.

It first uses the property element to restrict the input only from touch-tone keys. It then uses the field element to collect a digit input. The interface allows the user to specify a voice prompt, the speech error message when no input occurs, the speech help message when the user throws a help event, and the destination of the next dialog when the input is correctly done. Multiple inputs can be specified in one dialog. For example, for a login input dialog that inputs both account number and password, a simplified version of the generated VoiceXML file would like

```xml
<?xml version="1.0"?><vxml version="2.0">
<property name="inputmodes" value="tts"/>
<form id="login.vxml">
  <field name="var1" type="digits">
    <prompt bargein="true">please input your account number</prompt>
  </field>
  <field name="var2" type="digits">
    <prompt bargein="true">please input your password</prompt>
  </field>
  <filled>
    <submit next="login.php"
      namelist="var1 var2"/>
  </filled>
</form>
</vxml>

The voice_input dialog pattern uses the form element to input data by voice.

The interface allows the user to specify an SQL query to request the database for available prompts, the speech error message when no match occurs, the speech error message when no input occurs, the speech
help message when the user throws a help event, and the destination of the next dialog when the input is correctly done. Multiple inputs can be specified in one dialog. For example, a simplified version of the generated php file for a voice input would like

```php
$conn = @odbc_connect("host", "uid", "password");
if (!@$conn) {
    $sql = "Search SQL";
    $rs = @odbc_exec($conn, $sql);
    $recnum = odbc_num_rows($rs);
    if ($recnum == 0) {
        $vxml = $vxml . "<block>no data</block>\n";
    } else {
        $vxml = $vxml . "<field name="var1">\n";
        $vxml = $vxml . "<prompt bargein = \n";
        $vxml = $vxml . "<grammar>\n";
        $vxml = $vxml . "$myvar \n";
        $vxml = $vxml . "$vxml . "$myvar . "$var ;
    }
    odbc_close($conn);
    $vxml = $vxml . "$vxml . ":<grammar>\n";
    $vxml = $vxml . "$vxml . ":<filled>\n";
} else { handling database connection failure }
```

The search dialog pattern uses the data input from the key_input dialog or voice_input dialog to search the database and return the result to the next dialog.

The interface allows the user to specify the database and the data source to be queried, the SQL query, and the destinations of the next dialogs when the search succeeds or fails. For example, a simplified version of the generated php file for a database search would like

```php
$conn = @odbc_connect("host", "uid", "password");
if (!@$conn) {
    $sql = "Search SQL";
    $rs = @odbc_exec($conn, $sql);
    $recnum = odbc_num_rows($rs);
    if ($recnum == 0) {
        $vxml = $vxml . "<block>no data</block>\n";
        $vxml = $vxml . "Sorry No Data \n";
        $vxml = $vxml . "<goto next="success"\n";
        $vxml = $vxml . "$vxml .":<grammar>\n";
        $vxml = $vxml . "$myvar \n";
        $vxml = $vxml . ":<filled>\n";
        $vxml = $vxml . "$vxml .":<submit next = \n";
        $vxml = $vxml . "$vxml .":<grammar>\n";
        $vxml = $vxml . "$vxml .":<filled>\n";
        $vxml = $vxml . "$vxml .":<field>\n";
} else { handling database connection failure }
```

The verification dialog pattern uses the data input from the key_input dialog or voice_input dialog to verify the match with the information in the database.
The interface allows the user to specify the database and the data source to be verified, the SQL query, and the destinations of the next dialogs when the verification succeeds or fails. For example, for a login verification dialog that verifies both account number and password, a simplified version of the generated php file would like

```php
$conn=@odbc_connect("host", "uid", "passwd");
if (!$conn) {
    $sql= "Verifiy SQL";
    $rs=@odbc_exec($conn,$sql);
    $recnum=odbc_num_rows($rs);
    if ( $recnum == 0 ) {
        $vxml = $vxml . "<block>
        [Failur\e_Msg]
        <goto next="/failure" />
    } else {
        if ( $session_flag = true ) {
            $account = odbc_fetch_row($rs,account);
            if (!session_is_registered("cust_acc")) {
                session_register("cust_acc");
                $cust_acc=$account;
            }
        }
        $vxml = $vxml . "<goto next="/success" />
    } else { handling database connection failure }
}
else { handling database connection failure }
```

The update dialog pattern uses the data input from the key_input dialog or voice_input dialog to update information in the database.

```php
$conn=@odbc_connect("host", "uid", "passwd");
if (!$conn) {
    $sql= "[Update_SQL]";
    $rs=@odbc_exec($sql);
    if ( $rs ) $Err="Y";
    odbc_close();
    $vxml = $vxml . "<block>
    if ( $Err == "Y" )
        $vxml = $vxml . "[Success_Msg]"
    else
        $vxml = $vxml . "[Failure_Msg]"
    $vxml = $vxml . "<goto next="/success" />

    $vxml = $vxml . "</block>
```

The menu dialog pattern uses the menu element to allow the user to choosing amongst alternative destinations.
The interface allows the user to specify alternative functionalities and the corresponding dialog destinations, and the speech help message when the user throws a help event. For example, a simplified version of the generated VoiceXML file for a menu that provides two functionalities func1 and func2 would like

```xml
<menu id = "menuid" scope = "document">
  <prompt bargein = "true">
    please input func1 or func2
  </prompt>
  <choice dtmf = "1" next ="[dialog1]">
    func1
  </choice>
  <choice dtmf = "2" next ="[dialog2]">
    func2
  </choice>
</menu>
```

where dialog1 and dialog2 are the corresponding destination of func1 and func2.

5. Asynchronous dialog transfer events

The dialog design patterns are used to design dialogs and synchronous transfer among dialogs. This section describes the set of asynchronous transfer events among dialogs.

The TIVR tool provides eight predefined asynchronous dialog transfer events:

- Stop: terminate the current dialog and go to the previous menu dialog.
- Return: terminate the current dialog and go to the previous dialog.
- Repeat: repeat the current dialog.
- Help: present the help message to the user.
- First: go to the first page of a multiple-page data listing.
- Last: go to the last page of a multiple-page data listing.
- Previous: go to the previous page of a multiple-page data listing.
- Next: go to the next page of a multiple-page data listing.

These eight events are defined using the link element. For example, the repeat event is defined as follows:

```xml
<link event = "repeat">
  <grammar scope = document>
    \repeat
  </grammar>
</link>
```

Each event is scoped in the entire document. The voice used to throw an event is defined by the grammar element and is formed by prefixing the "backslash" to each event name to reduce the conflict with normal user input. The event handlers are implemented using the catch element. For example, the event handler of the repeat event is implemented as follows:

```xml
<catch event = "repeat">
  <reprompt />
</catch>
```

These eight events provide a standard mechanism to asynchronously transfer among dialogs.

6. Conclusions

This paper has described a development tool for VoiceXML-based interactive voice response systems. This tool provides a set of friendly graphical user interfaces to relieve users from writing code. This set of graphical user interfaces supports a set of commonly used dialog patterns to facilitate the design of the dialog structure of the system. This tool also provides a set of predefined asynchronous dialog transfer events to facilitate the transfer among dialogs.

This tool has been used to design a small bookstore application that provides functionalities for book information querying and book ordering. One of the main drawbacks of this tool is that it lacks a window to illustrate the overall structure of dialogs. Our future work will provide such a window. The TIVR is entirely based on English voice browser. Our future work will also develop a Chinese voice browser to facilitate the design of Chinese VoiceXML-based IVR systems.

References