Journal of Software Engineering Studies
Vol. 2, No. 2, 12-23, June 2007

Software Engineering Education in Taiwan

Voice Composer: A Development Tool for Voice Applications

Yi-Xuan Li, *Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan*
Nai-Wei Lin, *Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan*

Requests for tools supporting rapid development and maintenance of voice applications become urgent due to urgent requests for efficient and effective accesses of information and services via voice devices. This article introduces a development tool for voice applications that supports visual programming of voice applications. In particular, this tool addresses the extensibility of designing new dialog components and the integrity with database programming. These features further enhance the visual programming capabilities of previous development tools.

1.Introduction

Voice applications aim to support efficient and effective accesses of information and services via voice devices. Recently, personal mobile voice devices become prevalent. Thus, requests for efficient and effective accesses of information and services via personal mobile voice devices also become prevalent. Besides, voice applications also provide an easier way for eye-disabled people to efficient and effective accesses of information and services.

Voice applications share a similar property as web applications in that the information and services they provide are constantly changing. This makes the development and maintenance of voice applications and web applications difficult and tedious. Voice XML [4,8,9,10,16] provides a programming model for voice applications the same as HTML [7] for web applications. This programming model greatly facilitates the development and

maintenance of voice applications. A Voice XML gateway mainly incorporates the abilities of automatic speech recognition (ASR) and automatic synthesis of text to speech (TTS) to facilitate the interactions between web servers and users via voice devices as shown in Figure 1. Most voice applications are programmed in Voice XML.

A voice application consists of a collection of dialogs. Each dialog contains a succession of voice interactions between the user and the server. The control of the voice application transfers among the set of dialogs. The control may transfer from a dialog to another dialog and does not return, or the control may transfer from a dialog to a subdialog and later returns from that subdialog. As an example, in Figure 2, the control transfers from dialog D1 to dialog D2, and then to dialog D3. In dialog D2, the control transfers from D2 to subdialog SD1, and then to subdialog SD2. After the execution of subdialog SD2, the control transfers from SD2 back to SD1, and then back to D2.

Development tools for voice applications can be classified into three types. The first type of tools provides a Voice XML specific editor. Such tools include Voxeo Commuinity [12], BeVocal Café [2], Telera DevXchan [11], getVocal SDL [6], and others. The Voice XML editor aids to prompt the editing of correct Voice XML programs. The direct editing of Voice XML programs is however very time-consuming and tedious.

The second type of tools provides a set of wizards to automatically generate Voice XML codes for a set of basic dialog patterns. Most voice applications can be composed using dialogs generated from these basic dialog patterns. These tools significantly simplify the programming of Voice applications by avoiding programming directly in Voice XML. Such tools include VoiceGenie Developer Workshop [15], TIVR [3], and others. However, they still lack the visibility of the global structure of the application and the flexibility of rapidly maintaining the application.

The third type of tools provides visual programming of voice applications. These tools further simplify the programming and maintenance of voice applications. They provide graphic icons to represent basic dialog patterns, called dialog components.

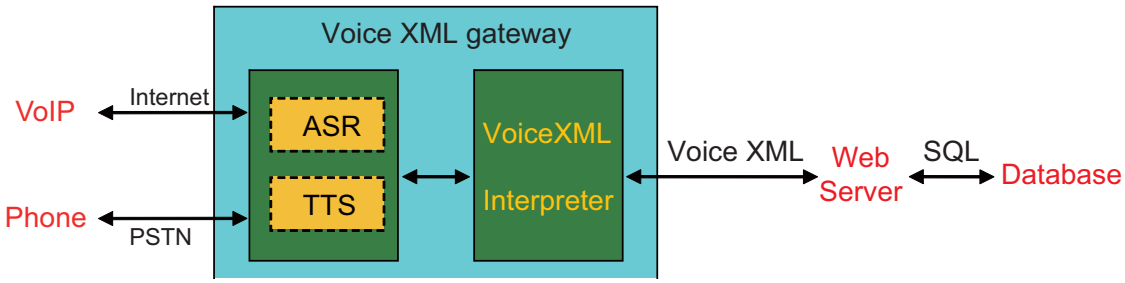

Figure 1. The architecture of voice applications.

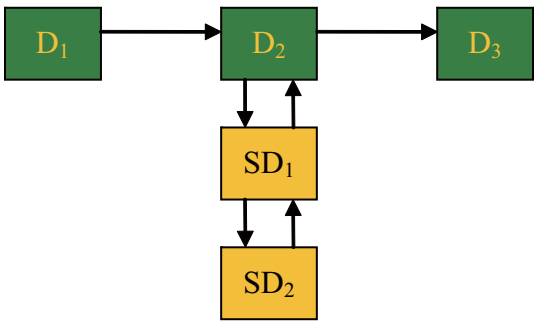

Figure 2. The control transfer among dialogs and subdialogs.

Users can directly drag and drop dialog components and connect them using links. These links represent the transfer of control between dialog components. Such a graphic notation, called a dialog flow, explicitly shows the global flow of control among dialog components in a voice application. This magnificently increases the visibility of the global structure of the application and the flexibility of maintaining the application. These tools then automatically generate Voice XML codes for these dialog flows. Such tools include Audium3 [1], V-Builder [13], Vocalocity [14], WebShere [17], and others. However, they still lack the extensibility of designing new dialog components and the integrity with database programming.

This article introduces a development tool for voice applications, called Voice Composer, which extends the third type of tools by addressing the extensibility of designing new dialog components and the integrity with database programming. This tool has been used to develop a voice web site for life education in a very short time. The contents of the voice web site can be updated easily and quickly.

The remainder of this article is organized as follows. Section 2 presents the architecture of Voice Composer. Sections 3 ~ 6 describe, respectively, the four main components of Voice Composer. Section 3 describes the Dialog Component Builder. Section 4 describes the Database Integrator. Section 5 describes the Dialog Flow Editor. Section 6 describes the Program Generator. Section 7 reports a case study to illustrate the effectiveness of Voice Composer and gives the conclusion of this article.

2. THE ARCHITECTURE OF VOICE COMPOSER

Voice Composer is designed to address the extensibility of designing new dialog components and the integrity with database programming. Voice Composer consists of a Project Manager, a Dialog Flow Editor, a Dialog Component Builder, a Database Integrator, a Program Generator and a Simulator. The architecture of Voice Composer is given in Figure 3.

A voice application consists of a set of structured dialog flows and a dialog flow consists of a set of structured dialog components. To develop a voice application project, the developer first uses the Project Manager to select the working directory of this project. All files of this project will be stored under its working directory. Each dialog flow will have a corresponding subdirectory under the working directory. The maintenance information about a project is stored in a .pj file.

Voice Composer provides a set of builtin dialog components. The developer can build new dialog components using the Dialog Component Builder if needed. These user-built dialog components can be reused and added into the set of builtin dialog components. This extensibility of designing new dialog components can incrementally enhance the development environment.

Most voice applications have extensive database accesses. The tedious programming of database connection can be greatly simplified using the Database Integrator. The Database Integrator is used to collect required information about database connections so that code for database connections can be generated automatically.

The Dialog Flow Editor is the main component for designing the dialog flows of a voice application using visual programming. Visual programming makes the development and maintenance of voice applications much more effective and efficient.

With the dialog flows of a voice application designed, the Program Generator will automatically generate Voice XML programs and associated PHP and SQL programs. This capability of automatic program generation makes the development and maintenance of voice applications much easier.

The Simulator can be used to verify the

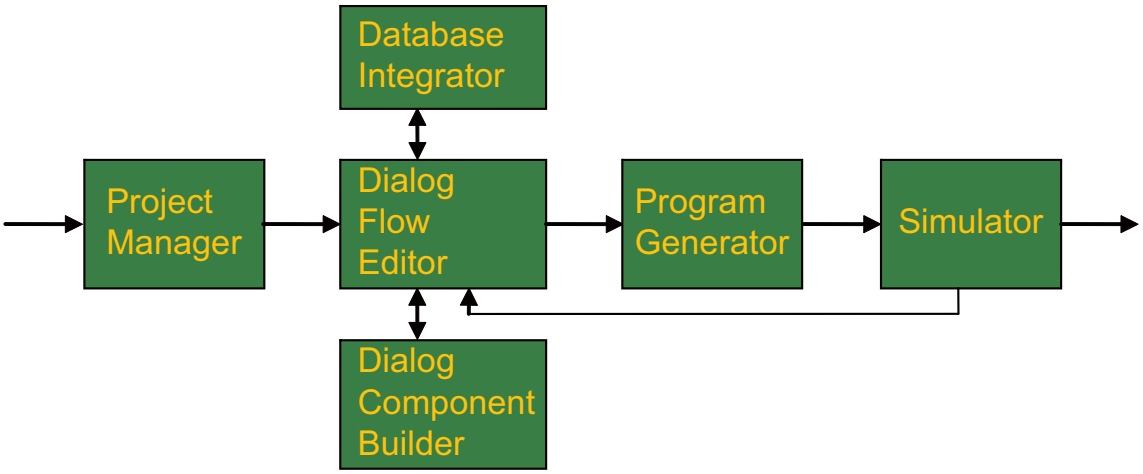

Figure 3. The architecture of Voice Composer.

functionalities of the voice application. If there are any errors, the developer can go back to the Dialog Flow Editor to correct the errors.

3: THE DIALOG COMPONENT BUILDER

Voice Composer has six builtin dialog components: start, exit, menu, link, SQL, and PHP. Each dialog flow in Voice Composer has a unique start component and a unique exit component. They are used to represent the entry and exit points of a dialog flow and have no attributes. The menu component provides a multiple choice control transfer via voice interactions. It has 2+2*n attributes for an n-choice menu. It has a choicenum attribute representing the number of choices in the menu, and a prompt attribute representing the string used to prompt the user's input. For each choice, it has a string attribute representing the matching user's input and a dtmf attribute representing the matching phone key. The link component provides direct control transfer to another dialog flow. It has a canvas attribute representing the name of a dialog flow. The SQL component is used to execute an SQL command. It has a command attribute representing the associated SQL command. The PHP component is used to execute a PHP program. It has a directory attribute representing the directory in which the PHP program resides.

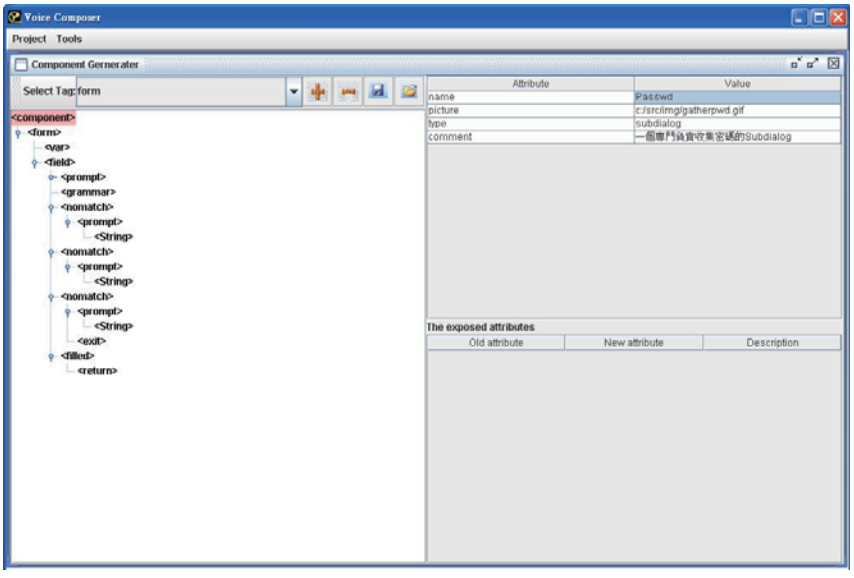

Figure 4. The Dialog Component Builder.

The Dialog Component Builder is used to build new dialog components. There are three types of dialog components: root dialog components, leaf dialog components, and subdialog components. The root dialog components are used to contain global variables and global events. A leaf dialog or a subdialog is used to execute a sequence of form elements or menu elements in Voice XML. When a subdialog is executed, it will transfer the control back to the dialog from which the control transfers.

The Dialog Component Builder is shown in Figure 4 and contains three windows. The left window shows the Voice XML tag tree structure of the dialog component. Initially, there is only one element with tag component. The component element contains four attributes. The attribute name is the name of this component. The attribute picture is the icon for this component. The attribute type is the type of this component. The attribute comment provides a description for this component.

To insert a child element e2 in the parent element e1, we first select the parent element e1, then select the child element e2 from the Select Tag window, and finally click the "+" button. For each selected parent element, only valid child elements can appear in the Select Tag window.

The up window in the right side shows the attributes of the selected element and their corresponding values. The designer can directly key in attribute values. An attribute is called an exposed attribute if its value should be provided by the user instead of the designer of the component. The down window in the right side shows the attributes exposed to the user. The name of an exposed attribute is contained in the old attribute column. The user can associate a new name for an exposed attribute in the new attribute column for better readability. The format of the attribute value can be described in the description column. The data structure for each dialog component consists of a component tree and a table for exposed attribute.

The Dialog Component Builder is usually used by developers who are familiar with Voice XML. The built dialog components can then be used by developers who are not familiar with Voice XML.

Consider the running example application in Figure 5. This application provides three functions in the main menu. A user can listen an essay, listen a music, or exit the application. After listening an essay or a music, the user can continue listening or exit the application. During listening, the user can press the '#' key to transfer to the main menu.

In this example application, we can use the builtin dialog component menu three times to implement the main menu, menu1, and menu2. We can design two new dialog components. One is used to implement the function of listening an essay, and another is used to implement the function of listening a music.

Using the Dialog Component Builder, we can design the dialog component for listening an essay as shown in Figure 6. We first set the attribute values of the component element. The attribute name is set to three-part. The attribute picture is set to the place where the graphic icon of this dialog component is stored. The attribute type is set to leaf. The three block elements are used to access the title, writer, and content of an essay. They contain three exposed attributes: title, writer, and content. The values of these exposed attributes will be set to SQL commands when the dialog component three-part is used to implement the function of listening an essay in the application. These SQL commands will then be executed to access the title, writer, and content of an essay in the database at runtime. We can similarly design the dialog component for listening a music.

We also need to design a new root dialog component, named simple_root, that can handle the global event of pressing the '#' key.

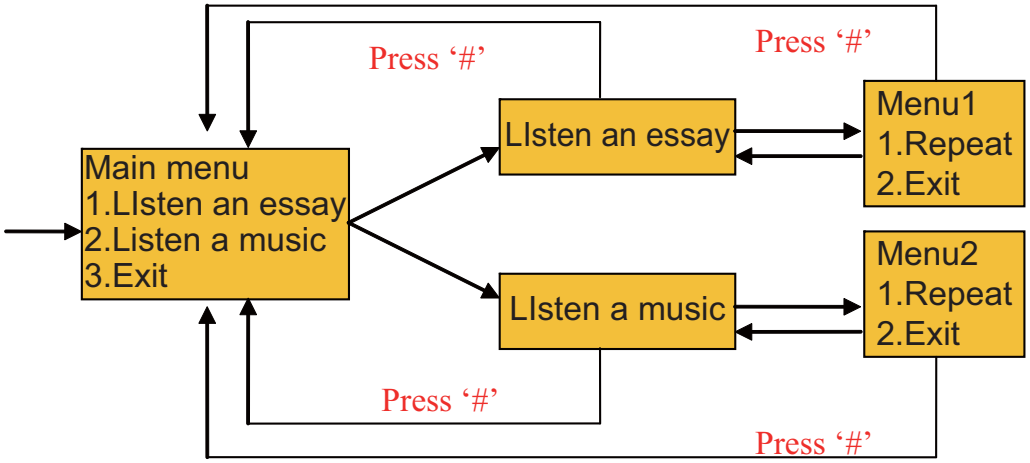

Figure 5. An example.

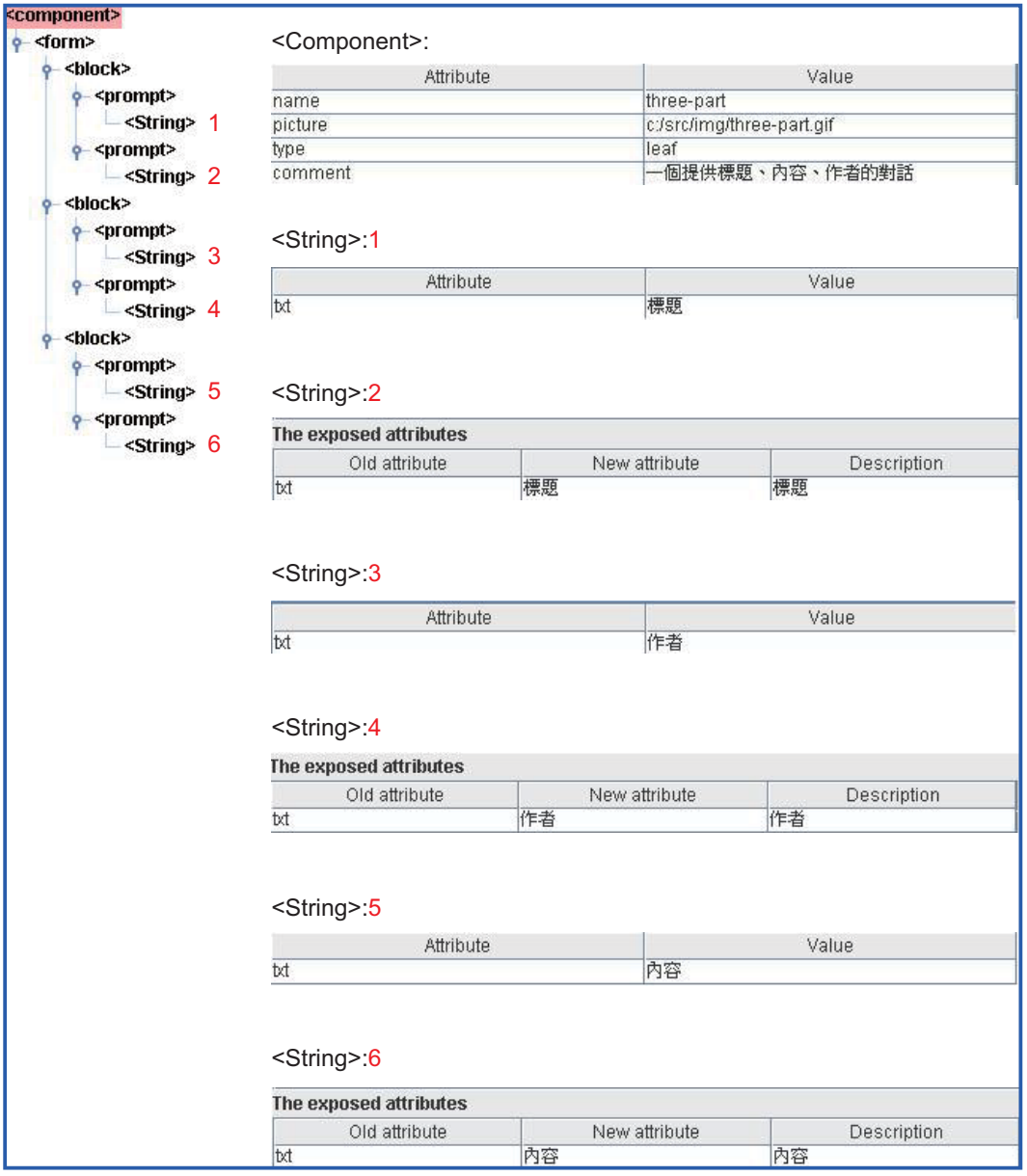

Figure 6. The dialog component for listening an essay.

4: THE DATABASE INTEGRATOR

The Database Integrator aims to simplify database programming. Voice Composer will generate code to access database via Open Database Connectivity (ODBC). ODBC is a standard software API for using SQL to access database independent of programming languages, database systems, and operating systems. The user first designs the schemas of a database. The Database Integrator is then used to specify the database connection information. The Database Integrator is shown in Figure 7. For each new database connection, there is a popup window asking for the information about the connection name, the database name, the user id, and the password. After obtaining the database connection information, the tables and fields in the database schemas are structured as a tree to facilitate the writing of SQL commands. With the database connection information and the database schemas information, the code for SQL and PHP components that access the database can be generated automatically.

For the running example above, assume that the essays are stored in the database named “test” and the connection is also named “test”. Then Figure 7 also shows the table and fields in the database “test”.

Figure 8 shows the dialog window for editing SQL commands. The user can select the connection name and the database name. After editing the core SQL command in the lower editing window and click the “√” button, the complete SQL command will appear in the upper editing window. Figure 8 shows an SQL command that accesses a database named “life” via a connection named “life”. The details for using this database connection information in database program generation are described in section 6.

5: THE DIALOG FLOW EDITOR

The Dialog Flow Editor is used to design the dialog flows of a voice application in visual programming. The Dialog Flow Editor is shown in Figure 9. The developer first selects the required dialog components in the dialog flow from the available dialog components in the left side of the window. If the required dialog component is not available, the developer can build the required dialog component using the Dialog Component

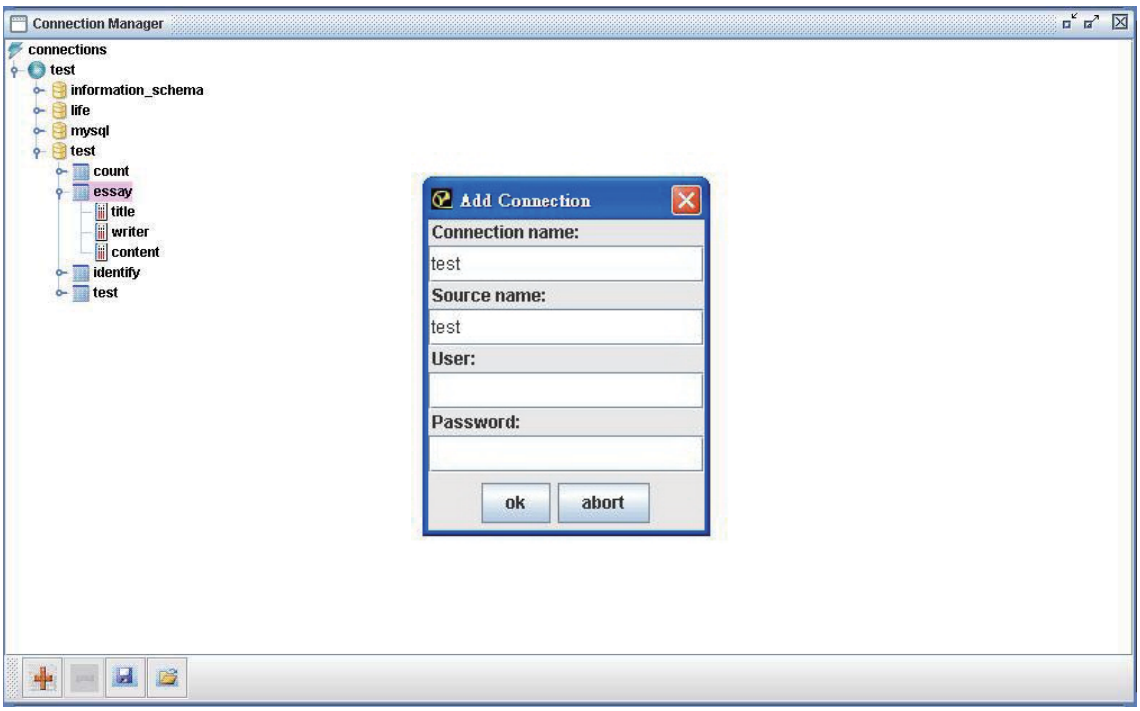

Figure 7. The Database Integrator.

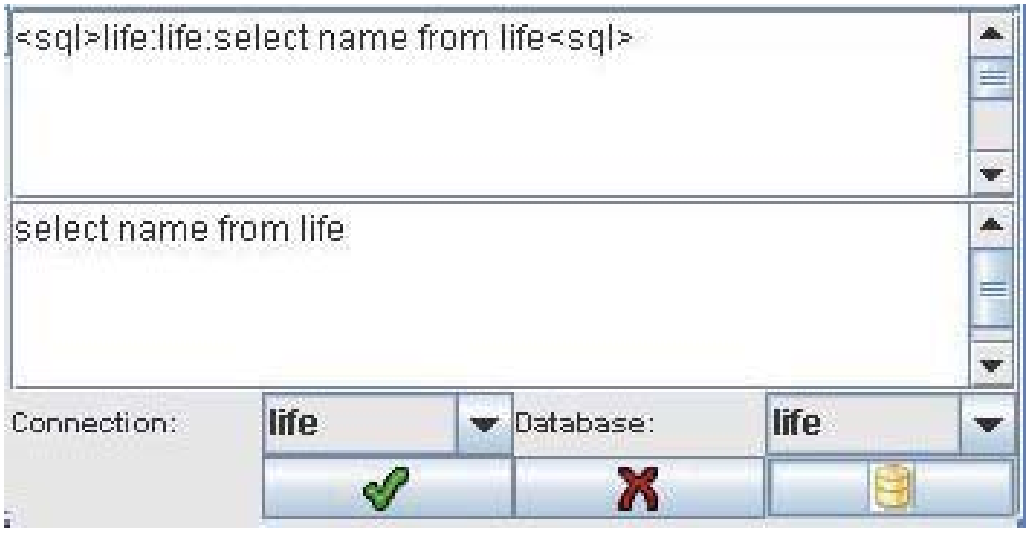

Figure 8. Dialog window for SQL command editing

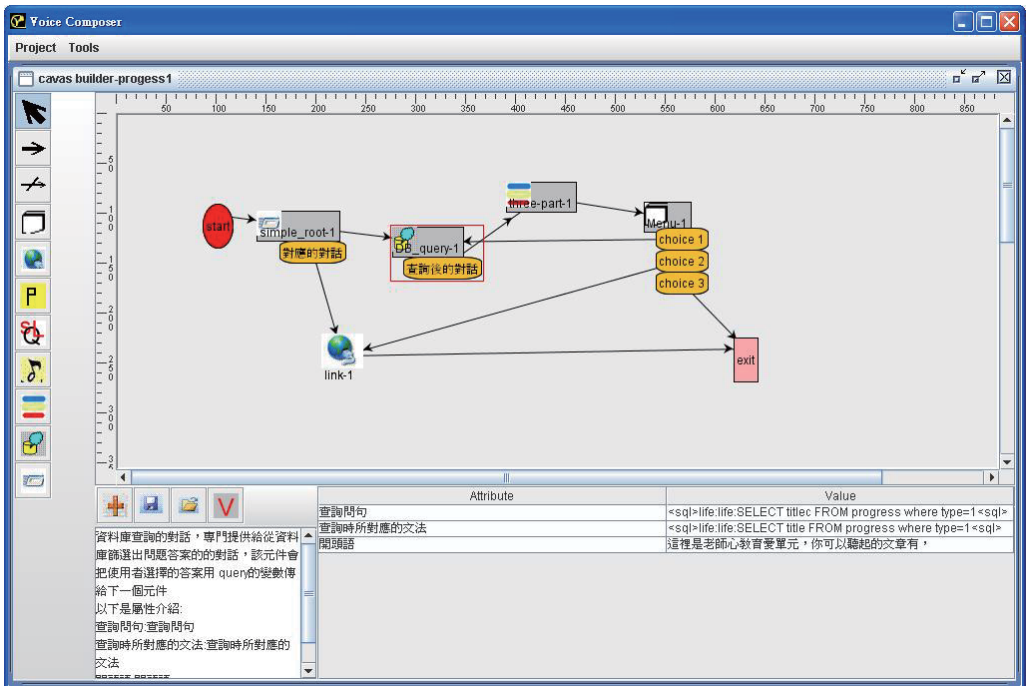

Figure 9. The Dialog Flow Editor.

Builder. The developer then connects the dialog components according to the flow of control among the dialog components. After each connection, the value for the corresponding control transfer attribute is set. At last, the developer sets the exposed attributes for each dialog component.

Figure 9 shows the dialog flow for listening an essay in the running example. The dialog flow will first load the root dialog simple_root-1 that contains the global event of pressing the "#" key. The dialog flow will then execute the three-part-1 dialog that reads an essay from the database and recites it. After the reciting, the dialog flow transfers to Menu-1 dialog. The user can then choose to either repeat the reciting or exit. If the user presses the "#' key at any time within the dialog flow, the control will transfer to the main menu via the link_1 component.

The Dialog Flow Editor will automatically verify whether a connection between two

components is consistent after a link is established. The Dialog Flow Editor will also automatically verify whether all the connections in a dialog flow are consistent whenever the developer stores the dialog flow. The verification is based on two types of information. First, the number of incoming edges and the number of outgoing edges for each dialog component are well-defined. For example, for the start component, the number of incoming edges must be zero and the number of outgoing edges must be one. For a 3-choice menu component, the number of incoming edges must be greater then the number of outgoing edges must be three. Second, some special restrictions must be satisfied between two kinds of dialog components. For example, a root dialog component can only be connected from the start component.

6: THE PROGRAM GENERATOR

The Program Generator will generate a Voice XML file or a PHP file for each dialog component. The program generation for builtin dialog components is very straightforward. For example, consider a menu dialog component with the following attribute values:

Attribute	Value
Choicenum	3
Menu_prompt	Select 1 for essay, select 2 for music, and select 3 to exit
String-1	Essay
dtmf-1	1
String-2	Music
dtmf-2	2
String-3	Exit
dtmf-3	3

The generated Voice XML file would be

```
<?xml version="1.0" encoding="Big5"?>
<Voice XML version="2.0" xmlns="http://
www.w3.org/2001/Voice XML">
<menu>
<prompt> Select 1 for essay, select
2 for music, and select 3 to exit</
prompt>
<choice dtmf="1" next="essay.xml"
>Essay</choice>
<choice dtmf="2" next="music.xml"
>Music</choice>
<choice dtmf="3" next="exit.xml">Exit</
choice>
</menu>
</Voice XML>
```

The program generation for user-defined dialog components is performed in three steps as shown in Figure 10. Each dialog component is represented as a component tree. To simplify the generation of the Voice XML programs, we use the available tool Xerces [18]. To use Xerces, we need to first convert our tree data structures into the DOM [5] tree data structures. In the second step, we use Xerces to convert the DOM tree data structures into Voice XML programs. If the dialog component involves PHP programs, we then convert the PHP programs in the third step.

In the first step, we perform a depth-first traversal of the component tree and convert nodes in the component tree into nodes in the DOM tree. An example of a component tree is shown in the left side of Figure 6 for the three-part dialog component. For most nodes in the tree, the conversion is very straightforward. We need to do special

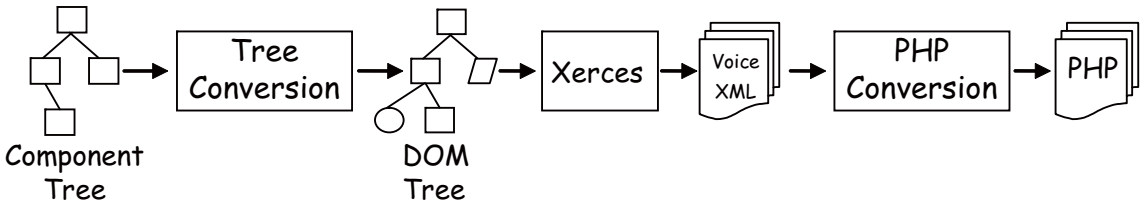

Figure 7. The Database Integrator.

handling for the following three types of nodes: grammars, SQL commands, and external variables. For grammars, a string may contain substrings separated by commas. For example, if the string for a grammar is “A, B, C”, then this string will be converted into

```
<one-of>
  <item>A</item>
  <item>B</item>
  <item>C</item>
</one-of>
```

Since each SQL command needs to establish the connection to the database, we need to add the connection procedure into the tree data structure. For example, the following SQL command

```
<sql>test:test:select question from
test<sql>
```

will be converted into

```
<?
$cnx = odbc_connect( 'test' , 'root',
'xxx' );
odbc_exec($cnx,"set NAMES 'big5'");
odbc_exec($cnx,'use test;');
$cur= odbc_exec( $cnx, "select question
from test" );
odbc_fetch_row( $cur ) ;
$value= odbc_result( $cur, 1 );
echo "$value";
odbc_close( $cnx);
?>
```

If the value of an attribute refers to an external variable x, the reference will be denoted as <v>$x</v>. This will be converted into the following PHP fragment <?echo “$x” ;?>.

In the second step, although there are many types of DOM tree nodes, we will use only three types of tree nodes: Element, Attr, and Text. In the third step, if the Voice XML file needs to converted into a PHP file, we need to handle “<?” and “?>” carefully since they are keywords for both Voice XML and PHP. For “<? … ?>” in Voice XML, we will convert it into “<?echo “<? … ?>”?>”. Therefore, this line

```
<?xml version='1.0' encoding='Big5'?>
```

will be converted into

```
<?echo “<?xml version='1.0' encoding='
Big5'?>” ?>
```

For “<?” and “?>”in PHP, we will first represent them as “<%” and “%>” in Voice XML file, and then convert them into “<?” and “?>”in PHP file. Therefore, the SQL command

```
<sql>test:test:select question from
test<sql>
```

will be first converted into

```
&lt;%
$cnx = odbc_connect( 'test' , 'root',
'xxx' );
odbc_exec($cnx,"set NAMES 'big5'");
odbc_exec($cnx,'use test;');
$cur= odbc_exec( $cnx, "select question
from test" );
odbc_fetch_row( $cur ) ;
$value= odbc_result( $cur, 1 );
echo "$value";
odbc_close( $cnx);
%&gt;
```

in the Voice XML file. It will then be converted into

```
<?
$cnx = odbc_connect( 'test' , 'root',
'xxx' );
odbc_exec($cnx,"set NAMES 'big5'");
odbc_exec($cnx,'use test;');
$cur= odbc_exec( $cnx, "select question
```

```
from test" );
odbc_fetch_row( $cur ) ;
$value= odbc_result( $cur, 1 );
echo "$value";
odbc_close( $cnx);
?>
```

in the PHP file.

For the three-part component shown in Figure 6, the generated PHP program is shown in Figure 11.

7: CONCLUSION

Voice Composer has been used in a project that converts part of a web site for life education into a voice web site for life education. The voice web site contains a large amount of articles, lectures, essays, and music. The cumulative time to complete this project is less than eight hours. It spent four hours on system analysis, two hours on database construction, one hour on building new components, and one hour on database integration and building dialog flows. This project contains 13 dialog flows and generates 124 Voice XML or PHP files. This project demonstrates that Voice Composer is a very efficient and effective tool for developing and maintaining voice applications.

```
<?echo "<?xml version='1.0' encoding='Big5'?>"?>
<vxml application="simple_root-1.vxml" version="2.0" xmlns="http://www.w3.org/2001/
vxml">
<form>
  <block>
    <prompt>標題</prompt>
    <prompt>
<?
$cnx = odbc_connect( 'test' , 'root', 'xxx' );
odbc_exec($cnx,"set NAMES 'big5'");
odbc_exec($cnx,'use test;');
$cur= odbc_exec( $cnx, "select title from essay where title='test'" );
while( odbc_fetch_row( $cur ) )
{
$value= odbc_result( $cur, 1 );
echo "$value,";
}
odbc_close( $cnx);
?>
</prompt>
  </block>
  <block>
    <prompt>作者</prompt>
    <prompt>
<?
$cnx = odbc_connect( 'test' , 'root', 'xxx' );
odbc_exec($cnx,"set NAMES 'big5'");
odbc_exec($cnx,'use test;');
$cur= odbc_exec( $cnx, "select writer from essay where title='test'" );
```

Figure 11. three-part-1.php.

Requests for tools supporting rapid development and maintenance of voice applications have become urgent due to urgent requests for efficient and effective accesses of information and services via voice devices. This article has introduced the development tool Voice Composer for voice applications that provides visual programming of dialog flows, and addresses the extensibility of designing new dialog components and the integrity with database programming.

ACKNOWLEDGEMENTS

This work was supported in part by the Ministry of Education of R.O.C. under the grant for the construction of a web site for life education. The authors would also like to thank the referees for many valuable comments and suggestions that greatly improve this article.

REFERENCES

[1] Audium3, http://www.audiumcorp.com/

[2] BeVocal Café, http://cafe.bevocal.com/

[3] C.-S. Chen, TIVR: A Development Tool For Interactive Voice Response, Master Thesis, Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan, R.O.C., 2002.

[4] Bob Edgar, The VoiceXML Handbook,CMP Books Press, 2001

[5] Document Object Model, http://www.w3.org/DOM/

[6] GetVocal, http://www.getvocal.com

[7] HTML, http://www.w3.org/TR/1999/REC-html401-19991224/

[8] IBM, Deep into VoiceXML, Part 1, Presented by developerWorks, ibm.com /developerWorks.

[9] IBM, Developing dynamic VoiceXML applications,Presented by developerWorks, ibm.com/developerWorks.

[10] Stephen Breitenbach ,Tyler Burd, Nirmal,Early Adopter VoiceXML,Wrox Press,Press,2001.

[11] Telera DeVXchan, http://www.telera.com

[12] Voxeo Community, http://community.voxeo.com

[13] V-Builder, http://support.nuance.com/

[14] Vocalocity, http://www.vocalocity.net/

[15] VoiceGenie, http://developer.voicegenie.com/

[16] VoiceXML, http://www.w3.org/TR/2004/REC-voicexml20-20040316/

[17] WebSphere, http://publib.boulder.ibm.com/infocenter/pvcvoice/51x/index.jsp

[18] XMLPaser, http://xerces.apache.org/

About the Author

Yi-Xuan Li

Yi-Xuan Li received M.S. degrees in Computer Science from National Chung Cheng University, Taiwan. He is currently an Software Engineer in the Department of Openfind Information Technology, Inc. His research interests include programming languages, compilers, XML based applications, and software tools.

Nai-Wei Lin

Nai-Wei Lin received the B.S. and M.S. degrees in Electrical Engineering from Tatung University, Taipei, R.O.C. in 1981 and 1983. He received the Ph.D. degree in Computer Science from University of Arizona, Tucson, Arizona, U.S.A. in 1993. He is currently an Associate Professor in the Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan. His research interests include programming languages, compilers, software testing, and software tools.