Voice Composer: A Development Tool for Voice Applications

Yi-Xuan Li and Nai-Wei Lin
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan 621, R.O.C.
{lyh93,naiwei}@cs.ccu.edu.tw

ABSTRACT

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 paper introduces a development tool for voice applications that provides visual programming of dialog components and dialog flows, and addresses the extensibility of dialog components and the integrity with database programming.

1: INTRODUCTION

Voice applications aim to support efficient and effective accesses of information and services via voice devices. Personal mobile voice devices recently become prevalent. Thus, requests for efficient and effective accesses of information and services via personal mobile voice devices also become prevalent. Furthermore, voice applications provide a critical 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 varied. This makes the maintenance of voice applications and web applications difficult and tedious. Voice XML [4,6,7,8,14,15] provides a programming model for voice applications the same as that for web applications. This programming model facilitates the maintenance of voice applications. In addition, Voice XML incorporates the abilities of speech recognition (ASR) and synthesis (TTS) to facilitate the interactions with users via voice user interface as shown in Figure 1. Currently, most voice applications are programmed in Voice XML.

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. Currently, there are three types of development tools for voice applications. The first type provides a Voice XML specific editor. Such tools include Voxeo Commuinity

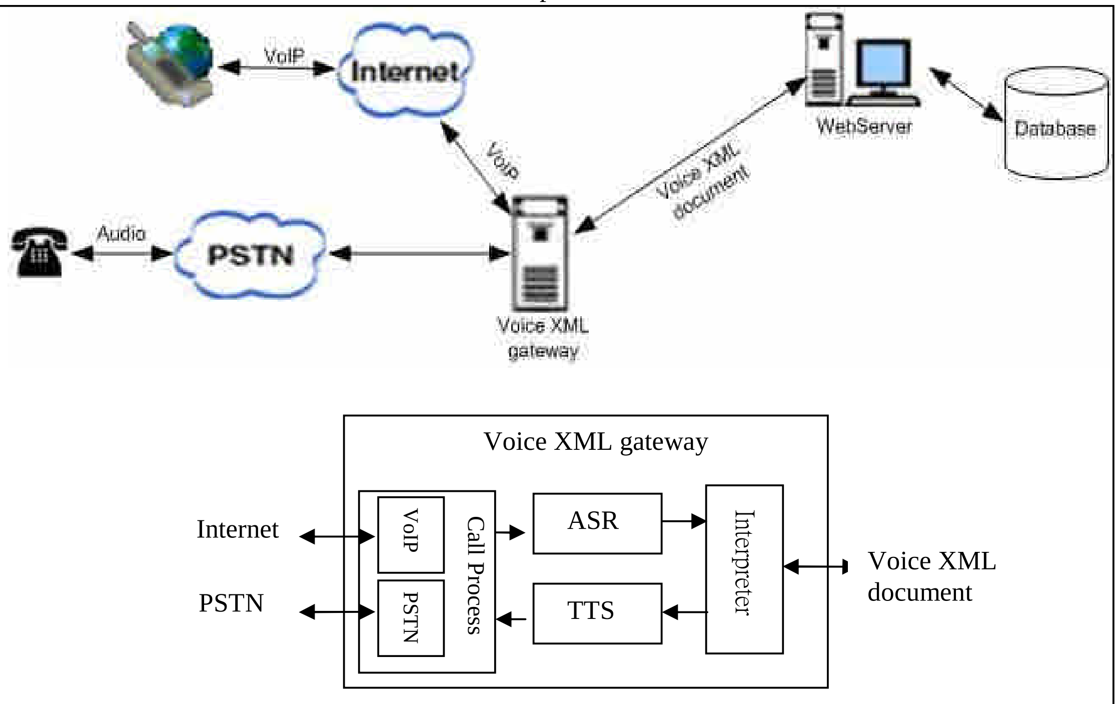


Figure 1. Voice XML application architecture.

[10], BeVocal Café [2], Telera DevXchan [9], getVocal SDL [5], and others. The editor aids to edit correct Voice XML. The direct editing of Voice XML is however more time-consuming and tedious.

The second type provides a set of wizards to facilitate the construction of some common dialog components. Such tools include VoiceGenie Developer Workshop [13], TIVR [3], and others. These tools significantly simplify the programming of Voice applications. The programmers do not have to do programming directly in Voice XML. However, they still lack the visibility of global structure of the application and the flexibility of maintaining the application.

The third type provides visual programming of dialog components and dialog flows. Such tools include Audium3 [1], V-Builder [11], Vocalocity [12], WebShere [16], and others. These tools further simplify the programming of voice applications. They can design the dialog flows of voice applications by dragging and dropping dialog components represented as graphic icons. This magnificently increases the visibility of global structure of the application and the flexibility of maintaining the application. However, they still lack the extensibility of dialog components and the integrity with database programming.

This paper introduces a development tool for voice applications, called Voice Composer, which addresses the extensibility of 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 content of the voice web site can also be updated quickly.

2: THE ARCHITECTURE OF VOICE COMPOSER

The tool Voice Composer is designed to address the extensibility of dialog components and the integrity with database programming. Voice Composers consists of 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 2. 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, the developer needs to design the required dialog components. Voice Composer provides a set of builtin dialog components. The developer can build new dialog components using 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 dialog components is very valuable. Most voice applications will have extensive database accesses. The tedious connection process to database systems can be simplified using the Database Integrator. This systematic integration of database programming into the tool is also very valuable.

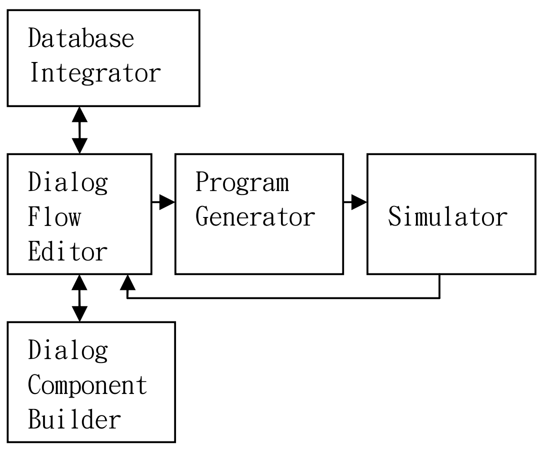


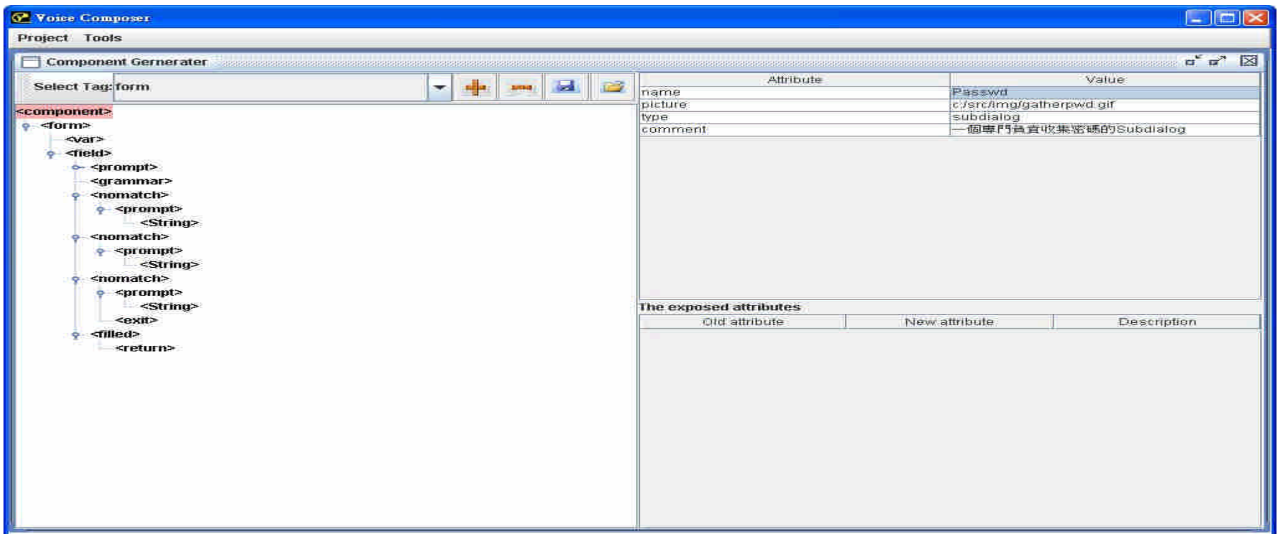
Figure 2. The architecture of Voice Composer.

The Dialog Flow Editor is the main component for design the dialog flows of a voice application in visual programming. The 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 and associated PHP and SQL programs. This capability of automatic program generation makes the maintenance of voice applications much easier. The Simulator can be used to verify the 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 input. For each choice, it has a string attribute representing the matching user 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 an 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.

The Dialog Component Builder can be used to build new dialog components. There are three types of dialog components: *root dialog* components, *leaf dialog* components, and *subdialog* components. The root



```
<component>
  <form>
    <var>
    <field>
      <prompt>
      <grammar>
      <nomatch>
        <prompt>
          <String>
      <nomatch>
        <prompt>
          <String>
      <nomatch>
        <prompt>
          <String>
        <exit>
      <filled>
        <return>
```

Attribute	Value
name	Passwd
picture	c:/src/img/gatherpwd.gif
type	subdialog
comment	一個專門負責收集密碼的Subdialog

Old attribute	New attribute	Description

Figure 3. The Dialog Component Builder.

dialog components are used to include 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 3 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.

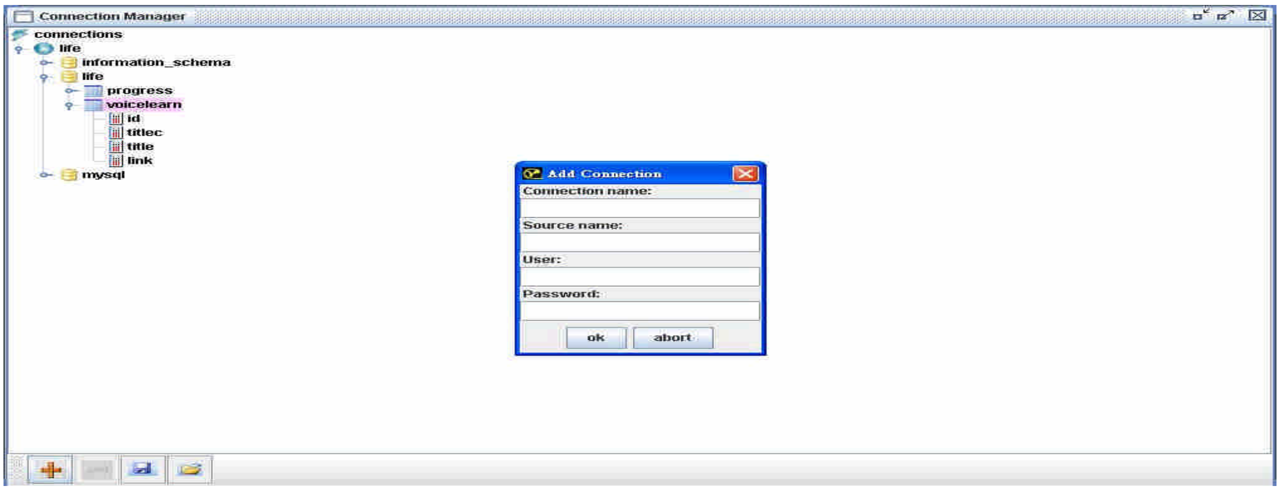
To insert a child element e_2 in the parent element e_1, we first select the parent element e_1, then select the child element e_2 from the Select Tag window, and finally click the "+" button. For each selected parent element, only valid child elements can appear in the Select Tag window.

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.

4: THE DATABASE INTEGRATOR

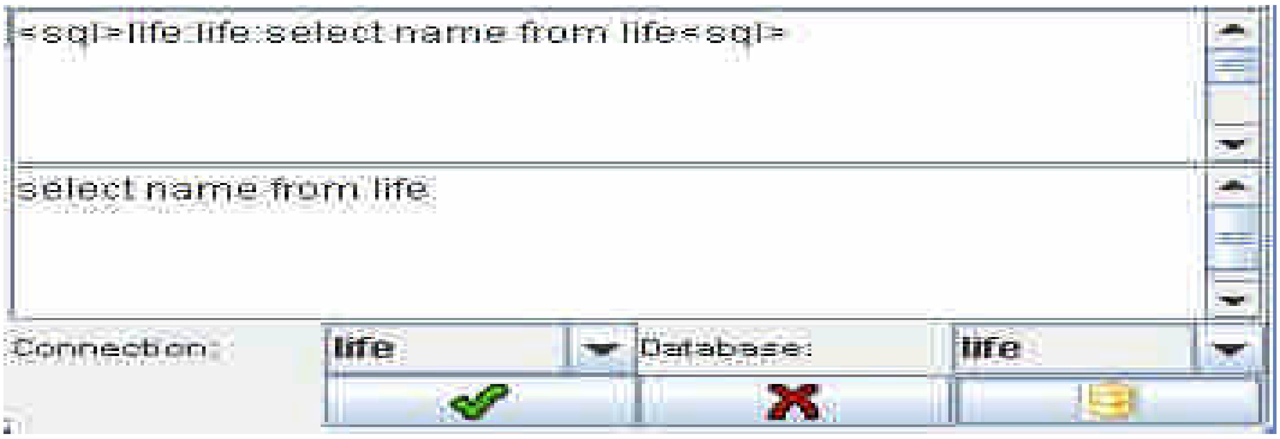
The Database Integrator is shown in Figure 4. The



```
connections
  life
    information_schema
    life
      progress
      voicelearn
        id
        titlec
        title
        link
    mysql
```

Connection name:	
Source name:	
User:	
Password:	

Figure 4. The Database Integrator.



```
<sql>life.life:select name from life<sql>
```

```
select name from life
```

Figure 5. Dialog window for SQL command editing

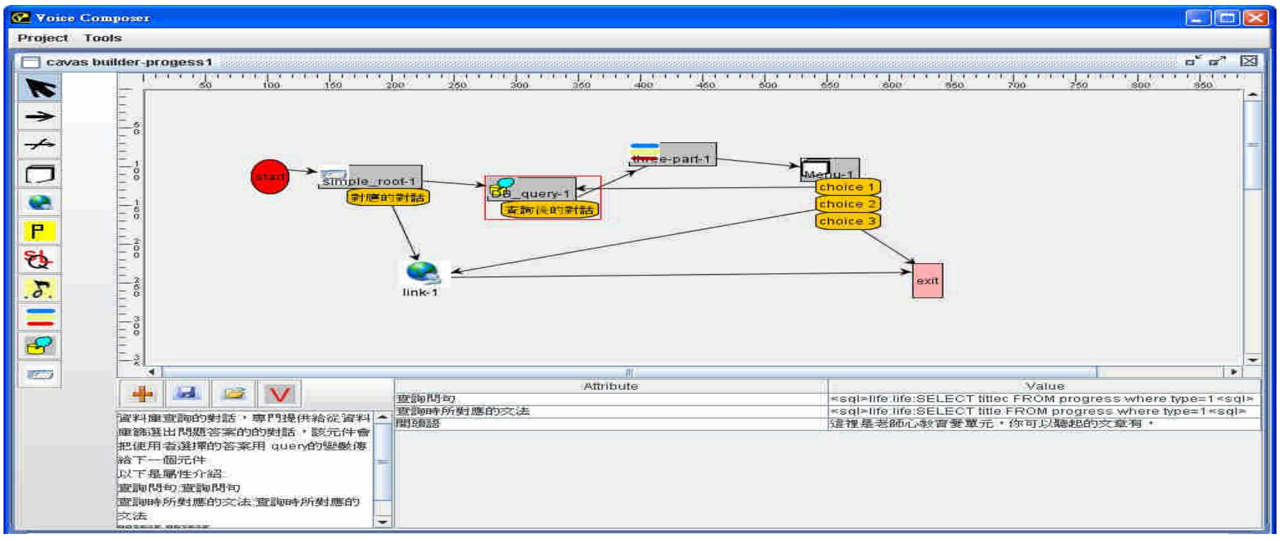
Database Integrator simplifies database programming by maintaining the database connection information and the database schemas information. For each new database connection, there is a popup window asking for the information about the connection name, the database name, the user id, and the password. With this information, SQL and PHP programs with database accesses can be generated automatically.

In addition to the database connection information, the tables and fields in the database schemas are structured as a tree to facilitate the writing of SQL commands. Figure 5 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. The details for using the database information in program generation are described in section 6.

5: THE DIALOG FLOW EDITOR

The Dialog Flow Editor designs the dialog flows of a voice application in visual programming. The Dialog Flow Editor is shown in Figure 6. 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 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.

The Dialog Flow Editor will automatically verify whether a connection is correct after the connection is established. The Dialog Flow Editor will also automatically verify whether all the connections are correct whenever the developer stores a dialog flow. The verification is based on the fact that the number of incoming edges and outgoing edges for each dialog component is known. For example, for the start component, the number of incoming edges is zero and the number of outgoing edges is one. For a 3-choice menu component, the number of incoming edges is



資料庫查詢的對話，專門提供給從資料庫篩選出問題答案的的對話，該元件會把使用者選擇的答案用 query的變數傳給下一個元件
以下是屬性介紹
查詢問句:查詢問句
查詢時所對應的文法:查詢時所對應的文法

Attribute	Value
查詢問句	<sql>life.life:SELECT titlec FROM progress where type=1<sql>
查詢時所對應的文法	<sql>life.life:SELECT title FROM progress where type=1<sql>
開頭語	這裡是老師心教育愛單元，你可以聽起的文章有，

Figure 6. The Dialog Flow Editor.

greater then one and the number of outgoing edges is three.

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, for a menu dialog component with the following attribute values:

Attribute	Value
Choicenum	3
Menu_prompt	Select 1 for article, select 2 for music, and select 3 to exit
String-1	Article
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 article, select 2 for music,
and select 3 to exit</prompt>
<choice dtmf="1"
next="article.xml">Article</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 7. Each dialog component is represented as a component tree. To simplify the generation of the Voice XML programs, we use the available tool Xerces [17]. To use Xerces, we need to first convert our tree data structures into the DOM 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. For most nodes in the tree, the conversion is very straightforward. We need to do special 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 process 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'
```

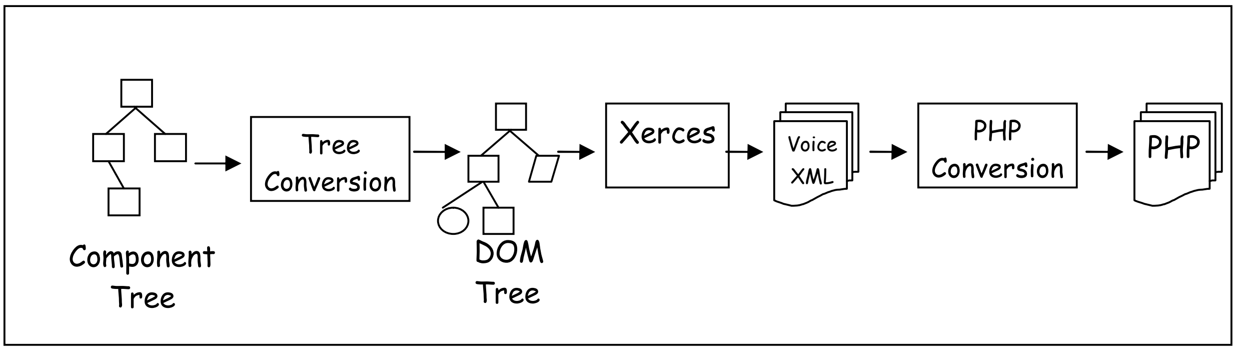


Figure 7. The structure of the Program Generator.

encoding=’Big5’?>” ?>
For “<?” and “?>”in PHP, we will first represent them as “&lt;%” and “%&gt;” 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.

7: CASE STUDY

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. This project is finished within 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 voice applications.

It should be noted that the maintenance of this voice web site will be very easy with the automatic program generation capability of Voice Composer.

8: CONCLUSION

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 paper has introduced the development tool Voice Composer for voice applications that provides visual programming of dialog components and dialog flows, and addresses the extensibility of 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 content of the voice web site can also be updated quickly.

9: 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 to improve this paper.

10: 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, National Chung Cheng University, 2002.
[4] Bob Edgar, The VoiceXML Handbook,CMP Books Press, 2001
[5] GetVocal, http://www.getvocal.com
[6] IBM, Deep into VoiceXML, Part 1, Presented by developerWorks, ibm.com /developerWorks.
[7] IBM, Developing dynamic VoiceXML applications,Presented by developerWorks, ibm.com/developerWorks.
[8] Stephen Breitenbach ,Tyler Burd, Nirmal,Early Adopter VoiceXML,Wrox Press,Press,2001.
[9] Telera DeVXchan, http://www.telera.com
[10] Voxeo Community, http://community.voxeo.com
[11] V-Builder, http://support.nuance.com/
[12] Vocalocity, http://www.vocalocity.net/
[13] VoiceGenie, http://developer.voicegenie.com/
[14] Voice XML, http://www.w3.org/DOM/
[15] W3C, Extensible Markup Language (VoiceXML) Version 2.0, Recommendation 16 March 2004. http://www.w3.org/TR/2004/REC-voicexml20-20040316/
[16] WebSphere, http://publib.boulder.ibm.com/infocenter/pvcvoice/51x/index.jsp
[17] XMLPaser, http://xerces.apache.org/