Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2493533 C 2012/06/12

(11)(21) 2 493 533

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(86) Date de dépôt PCT/PCT Filing Date: 2003/07/24
(87) Date publication PCT/PCT Publication Date: 2004/01/29
(45) Date de délivrance/Issue Date: 2012/06/12
(85) Entrée phase nationale/National Entry: 2005/01/20
(86) N° demande PCT/PCT Application No.: AU 2003/000939
(87) N° publication PCT/PCT Publication No.: 2004/010678
(30) Priorité/Priority: 2002/07/24 (AU2002950336)

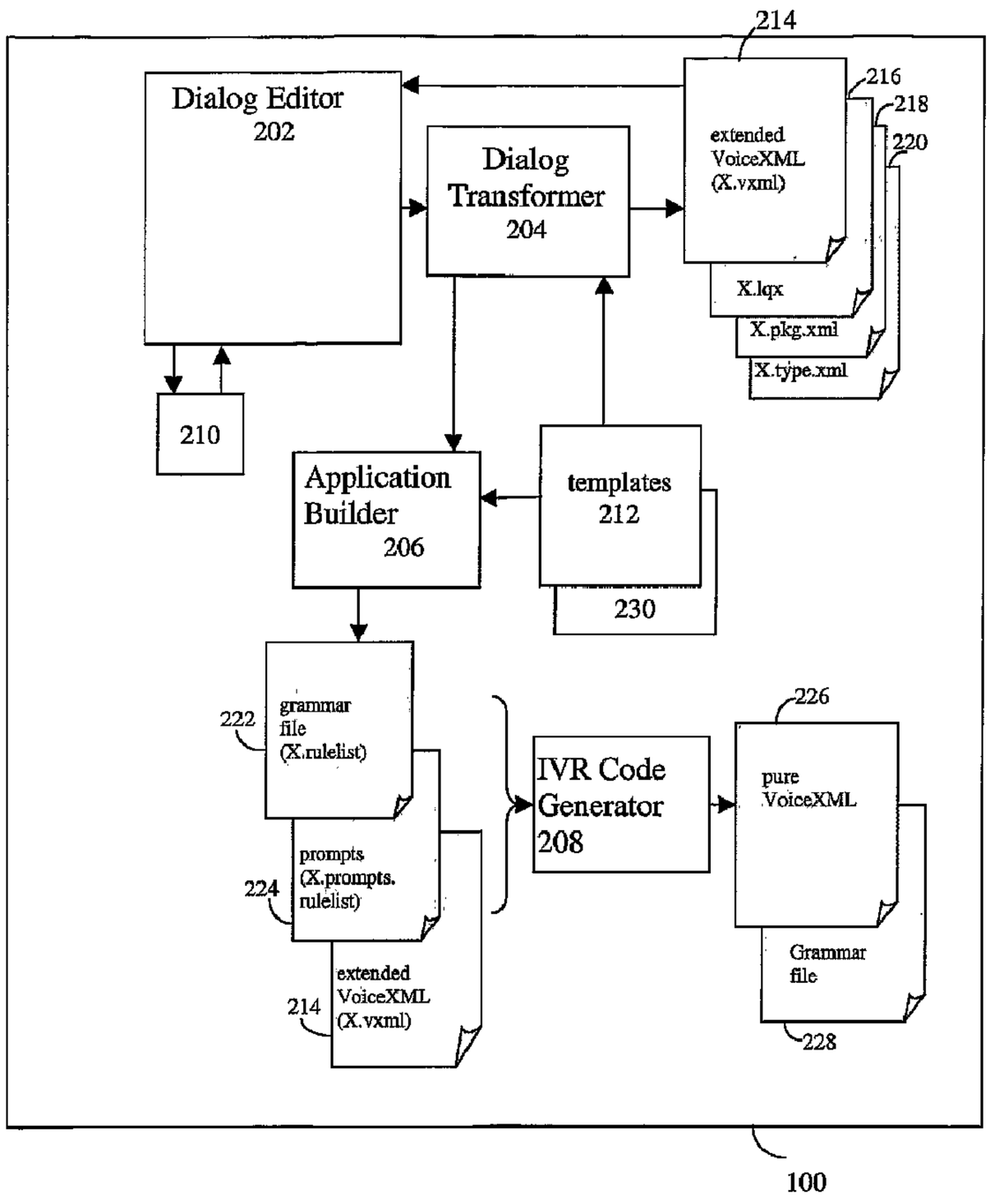
(51) Cl.Int./Int.Cl. *H04M 3/493* (2006.01), *G06F 17/20* (2006.01), *G10L 15/28* (2006.01), *H04M 1/64* (2006.01)

(72) Inventeurs/Inventors:
LAW, ENG BOON, AU;
HO, KHANH THI PHUONG, AU;
HUI, ALVARO KAU KAM, AU;
STARKIE, BRADFORD CRAIG, AU

(73) Propriétaire/Owner:
TELSTRA CORPORATION LIMITED, AU

(74) Agent: SMART & BIGGAR

(54) Titre : SYSTEME ET PROCEDE DE DEVELOPPEMENT D'UNE APPLICATION VOCALE
(54) Title: SYSTEM AND PROCESS FOR DEVELOPING A VOICE APPLICATION



(57) Abrégé/Abstract:
A system for use in developing a voice application, including a dialog element selector for defining execution paths of the application by selecting dialog elements and adding the dialog elements to a tree structure, each path through the tree structure

CA 2493533 C 2012/06/12

(11)(21) 2 493 533

(13) C

(57) Abrégé(suite)/Abstract(continued):
representing one of the execution paths, a dialog element generator for generating the dialog elements on the basis of predetermined templates and properties of the dialog elements, the properties received from a user of the system, each of said dialog elements corresponding to at least one voice language template, and a code generator for generating at least one voice language module for the application on the basis of said at least one voice language template and said properties. The voice language templates include VoiceXML elements, and the dialog elements can be regenerated from the voice language module. The voice language module can be used to provide the voice application for an IVR.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 January 2004 (29.01.2004)

PCT

(10) International Publication Number
WO 2004/010678 A1

(51) International Patent Classification[7]: H04M 3/493, 1/64, G06F 17/20, 19/00

(21) International Application Number: PCT/AU2003/000939

(22) International Filing Date: 24 July 2003 (24.07.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
2002950336 24 July 2002 (24.07.2002) AU

(71) Applicant *(for all designated States except US)*: **TELSTRA NEW WAVE PTY LTD** [AU/AU]; 242 Exhibition Street, MELBOURNE, Victoria 3000 (AU).

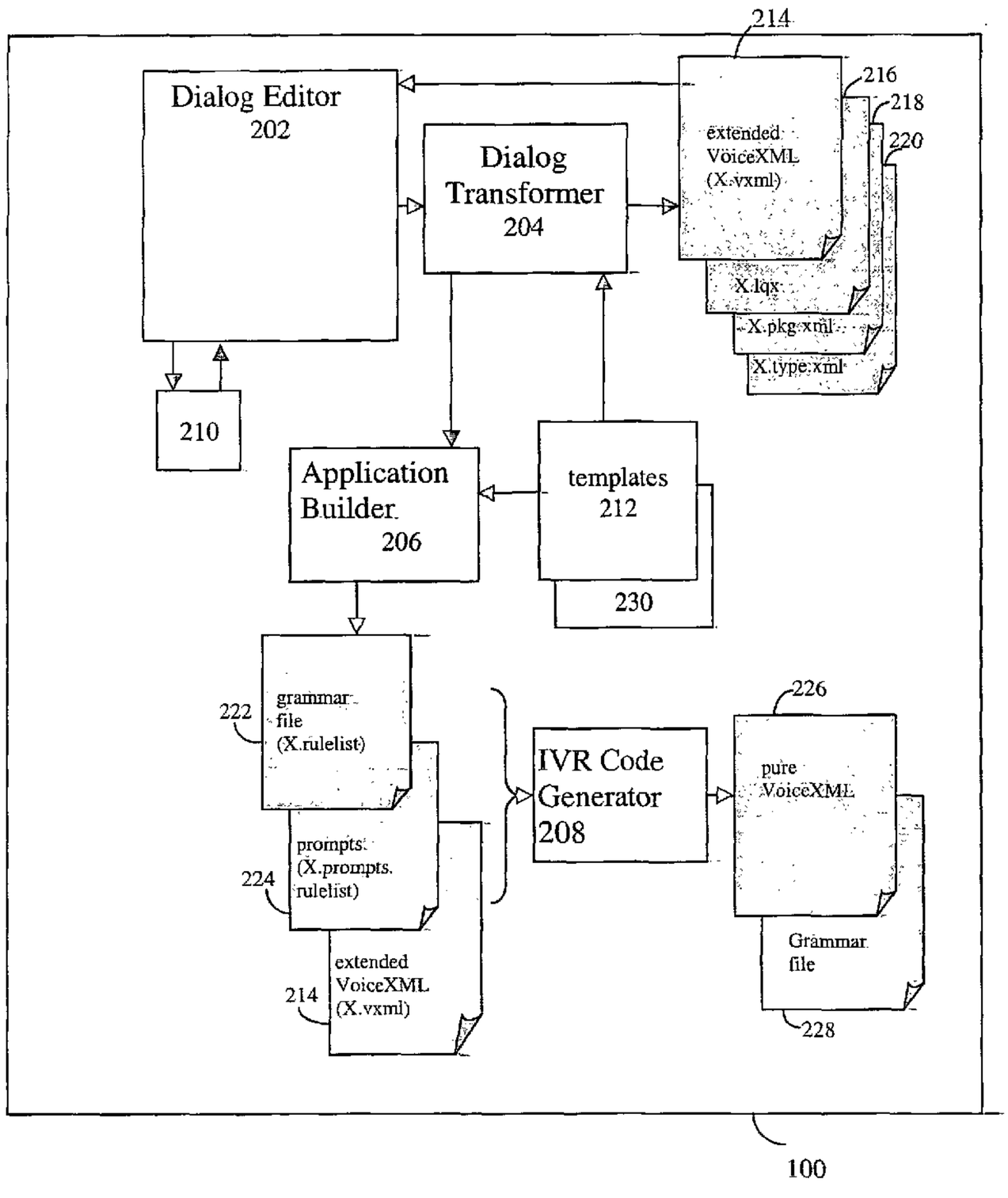
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **LAW, Eng, Boon** [MY/AU]; 7 Ardene Court, HAWTHORN, Victoria 3122 (AU). **HO, Khanh, Thi, Phuong** [AU/AU]; Unit 4, 900 Station Street, BOX HILL NORTH, Victoria 3129 (AU). **HUI, Alvaro, Kau, Kam** [AU/AU]; 18 Kelvinside Drive, TEMPLESTOWE, Victoria 3106 (AU). **STARKIE, Bradford, Craig** [AU/AU]; 52 Boisdale Street, SURREY HILLS, Victoria 3127 (AU).

(74) Agents: **WEBBER, David, Brian** et al.; DAVIES COLLISON CAVE, 1 Little Collins Street, MELBOURNE, Victoria 3000 (AU).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

[Continued on next page]

(54) Title: SYSTEM AND PROCESS FOR DEVELOPING A VOICE APPLICATION



(57) Abstract: A system for use in developing a voice application, including a dialog element selector for defining execution paths of the application by selecting dialog elements and adding the dialog elements to a tree structure, each path through the tree structure representing one of the execution paths, a dialog element generator for generating the dialog elements on the basis of predetermined templates and properties of the dialog elements, the properties received from a user of the system, each of said dialog elements corresponding to at least one voice language template, and a code generator for generating at least one voice language module for the application on the basis of said at least one voice language template and said properties. The voice language templates include VoiceXML elements, and the dialog elements can be regenerated from the voice language module. The voice language module can be used to provide the voice application for an IVR.

WO 2004/010678 A1

WO 2004/010678 A1

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

66718-76

SYSTEM AND PROCESS FOR DEVELOPING A VOICE APPLICATION

FIELD OF THE INVENTION

The present invention relates to a system and process for generating a voice application.

BACKGROUND

A voice application is a software application that provides an interactive audio interface, particularly a speech interface, on a machine, such as an Interactive Voice Response (IVR) system. IVRs, such as Intel's Dialogic™ IVR, are used in communications networks to receive voice calls from parties. The IVR is able to generate and send voice prompts to a party and receive and interpret the party's responses made in reply.

Voice extensible markup language, or VoiceXML, is a markup language for voice or speech-driven applications. VoiceXML is used for developing speech-based telephony applications, and also enables web-based content to be accessed via voice using a telephone. VoiceXML is being developed by the VoiceXML Forum, and is described at http://www.voicexml.org. Due to the verbose nature of VoiceXML, it can be cumbersome to develop VoiceXML-based applications manually using a text or XML editor. Consequently, voice application development systems are available that allow voice applications to be developed by manipulating graphical elements via a graphical user interface rather than coding VoiceXML directly. However, these systems are limited in their ability to assist a developer. It is desired to provide a process and system for developing a voice application that improves upon the prior art, or at least provide a useful alternative to existing voice application development systems and processes.

SUMMARY OF THE INVENTION

In accordance with some embodiments of the present invention, there is provided a process for developing a voice application, including:

generating graphical user interface components for defining execution paths of said application by arranging dialog elements in a tree structure, each path through said tree structure representing one of said execution paths;

generating said dialog elements on the basis of predetermined templates and properties of said dialog elements, said properties received from a user via said graphical user interface components, each of said dialog elements corresponding to at least one voice language template; and

generating at least one voice language module for said application on the basis of said at least one voice language template and said properties.

Some embodiments of the present invention also provide a system for use in developing a voice application, including:

a dialog element selector for defining execution paths of said application by selecting dialog elements and adding said dialog elements to a tree structure, each path through said tree structure representing one of said execution paths;

a dialog element generator for generating said dialog elements on the basis of predetermined templates and properties of said dialog elements, said properties received from a user of said system, each of said dialog elements corresponding to at least one voice language template; and

a code generator for generating at least one voice language module for said application on the basis of said at least one voice language template and said properties.

Some embodiments of the present invention also provide a graphical user interface for use in developing a voice application, said interface including graphical user interface components for defining execution paths of said application by arranging configurable dialog elements in a tree structure, each path through said tree structure

representing one of said execution paths, and said dialog element components may include one or more of:

a start dialog component for defining the start of said application;

a variables component for use in defining variables for said application;

a menu component for defining a menu;

a menu choice component for defining a choice of said menu;

a decision component for defining a decision branching point;

a decision branch component for defining a test condition and an execution branch of said decision branching point;

a form component for defining a form to collect input from a caller;

a record component for recording audio

a speaker component for playing prompts,

a local processing component for defining local processing;

a remote processing component for performing processing on a remote system;

a loop component for defining an execution loop;

a loop call component for calling said loop;

a loop next component for proceeding to the next cycle of said loop;

a loop break component for breaking out of said loop;

a subroutine component for defining a subroutine;

a subroutine call component for calling said subroutine;

subroutine return component for returning from said subroutine;

a jump component for defining a non-sequential execution path to a dialog element

a transfer component representing the transfer of a call to another number

a hotwords component for defining a word or phrase and a non-sequential execution path to a dialog element to be followed upon receipt of said word or phrase; and

an end component for defining an end of said application.

Some embodiments of the present invention also provide a process for developing a voice application, including: generating graphical user interface components for selecting and interconnecting graphical icons representing dialog elements for voice applications, said dialog elements representing functional components of voice applications, having user configurable properties, and corresponding to respective predetermined sequences of VoiceXML elements; receiving user input generated by user interaction with said graphical user interface components, said user input representing selection, arrangement, and interconnections of ones of said graphical icons, and configuration of dialog elements corresponding to ones of the selected graphical icons; processing said user input to define a voice application by selecting dialog elements corresponding to the selected ones of said icons, configuring properties of ones of the selected dialog elements, and defining execution flows of said voice application from the interconnections of the selected icons; and generating application code to provide said voice application; wherein the selected icons are interconnected to define a tree structure of execution flows for the voice application, each path through said tree structure representing a corresponding one of said execution flows; and said application code represents each dialog element of said application as a linked sequence of VoiceXML elements including extended attributes

that allows the tree structure of execution flows to be determined from said application code.

Some embodiments of the present invention also provide a system having components for executing a process as described in the paragraph above.

Some embodiments of the present invention also provide a computer readable storage medium having stored thereon a voice application generated by executing a process as described above.

Some embodiments of the present invention also provide a computer readable storage medium having stored thereon executable programming instructions that, when executed by at least one computer processor, cause the processor to execute a process as described above.

Some embodiments of the present invention also provide a system for use in developing a voice application, including: a dialog element selector for defining execution flows of a voice application by selecting and interconnecting graphical icons representing dialog elements for said voice application, said dialog elements representing functional components of said voice application, having user configurable properties, and corresponding to respective predetermined sequences of VoiceXML elements; means for receiving user input generated by user interaction with said dialog element selector, said user input representing selection, arrangement, and interconnections of ones of said graphical icons, and configuration of dialog elements corresponding to ones of the selected graphical icons; means for processing said user input to define a voice application by selecting dialog elements corresponding to the selected ones of said icons, configuring properties of ones of the selected dialog elements, and defining execution flows of said voice application from the interconnections of said icons; and a code generator for generating application code to provide said voice application; wherein the interconnections of the selected icons define a tree structure of execution flows for the voice application, each path through said tree structure representing a corresponding one of said execution flows; and said application code represents each dialog element of said

application as a linked sequence of VoiceXML elements including one or more extended attributes that allow the tree structure of execution flows to be determined from said application code.

Some embodiments of the present invention also provide a computer readable storage medium having stored thereon an extended VoiceXML file generated by a system as described in the paragraph above.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention are hereinafter described, by way of example only, with reference to the accompanying drawings, wherein:

Figure 1 is a block diagram showing a preferred embodiment of a voice application development system connected to an IVR via a network, and a telephone connected to the IVR via the PSTN;

Figure 2 is a schematic diagram of the voice application development system, showing how a voice application is developed;

Figure 3 is a flow diagram of a voice application development process executed by the system;

Figure 4 is a screenshot of a graphical user interface of the voice application development system;

Figure 5 is a screenshot of a dialog element selection bar of the graphical user interface; and

Figure 6 is a flow diagram of a code generation process executed by the system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

As shown in Figure 1, a voice application development system 100 can be connected to a VoiceXML-enabled interactive voice response system (IVR) 102 via a communications network 104. The system 100 executes a voice application development process which allows an application developer to develop a speech based application using a graphical user interface of the system 100. The application can then be transferred to the IVR 102 via the network 104. A standard telephone 106 can be used to access the IVR 102 via the public switched telephone network (PSTN) 108, allowing a user of the telephone 106 to interact with the speech based application simply by speaking into the telephone 106 to provide speech input to the application in response to voice prompts provided by the IVR 102 from the application. In the described embodiment, the voice application development system 100 is a standard computer system, such as an Intel™-based personal computer running a Microsoft Windows™ operating system, and the voice application development process is implemented by software modules stored on hard disk storage of the voice application development system 100. However, it will be apparent to those skilled in the art that at least parts of the voice application development process can be alternatively implemented by dedicated hardware components such as application-specific integrated circuits (ASICs). The voice application runs on the IVR 102, which may be an Intel Dialogic™ IVR with Nuance's Voice Web Server™ software. The network 104 may be any secure communications network that enables voice applications to be loaded onto the IVR 102, such as an Ethernet LAN or TCP/IP network.

As shown in Figure 2, the voice application development system 100 includes a dialog

editor module 202, a dialog transformer module 204, an application builder module 206, an IVR code generator 208, other application development modules 210, VoiceXML templates 212, and EMCAscript templates 230. The voice application development system 100 constitutes an integrated development environment (IDE) for the development of speech based applications. The system 100 executes an application development process, as shown in Figure 3, that allows a user of the system 100 to develop a voice application for a particular IVR platform.

When the process begins, the system 100 generates a graphical user interface, as shown in Figure 4. The interface is in the form of a window 400 with a project pane 402, a tools pane 404, and a messages pane 406. The window 400 also includes a main menubar 408 and a main toolbar 410. The main menubar 408 includes a Tools menu that provides access to a number of modules of the system 100 that are used to develop voice applications, as described below, and the tools pane 404 provides an interface to each tool when that tool is executed.

To develop a speech based application, a user of the system 100 can create a new project or open a saved project by selecting a corresponding menu item from the "Files" menu of the main menubar 408. The dialog editor 202 is then executed, and a tabbed dialog panel 411 is added to the tools pane 404, providing an interface to the dialog editor 202, and allowing the user to define an execution flow, referred to as a dialog, for the application. The dialog panel 411 includes a dialog pane 412, a dialog element toolbar 414 referred to as the dialog palette, a dialog element properties pane 416, and a dialog element help pane 418.

An application can be built from a set of seventeen dialog elements represented by icons in the dialog palette 414, as shown in Figures 4 and 5. Each element represents a complete or partial component of the voice application, such as a menu, a menu choice, a form, an execution loop, a speech prompt, and so on. The full set of dialog components is given in Appendix A. A dialog element is added to the dialog by selecting the element from the dialog palette 414 using a pointing device of the system 100 such as a mouse or tablet, and

placing the selected dialog element on the dialog editor pane 412 using a drag-and-drop action. Each dialog element has a number of properties that can be set by the user. Once placed in the dialog editor pane 412, an instance of the selected dialog element is added to the dialog and its properties can be set. When a dialog element instance in the dialog editor pane 412 is selected, its property names and associated values are displayed in the properties pane 416. The properties pane displays the name of each property, and includes controls such as check boxes and buttons to allow the user to modify the values of existing properties, and to add or delete new properties. The dialog element help pane 418 displays help information for the selected element, facilitating the rapid development of the application.

The execution flow of the application is defined by adding dialog elements to the dialog editor pane 412, setting the properties of the dialog elements, and defining the execution order of the dialog elements. The latter is achieved by dragging a dialog element and dropping it on top of an existing dialog element in the dialog editor pane 412. The dropped element becomes the next element to be executed after the element that it was dropped onto. The sequence and properties of dialog elements on the dialog editor pane 412 defines a dialog. Thus a dialog represents the execution flow of a voice application as a sequence of dialog elements. This sequence represents the main flow of the application and provides a higher-level logical view of the application that is not readily evident from the application's VoiceXML code. Thus the dialog provides a clear and logical view of the execution of the application. In addition to the main flow, non-sequential execution branches can be created by using a Jump dialog element. However, such non-sequential execution is not represented in a dialog. A subroutine is represented by an icon in the project pane 402 and appears as an icon in the dialog editor pane 412 when the main dialog is displayed. The execution flow of a subroutine can be displayed by selecting its icon in the project pane 402.

The sequencing of a dialog is facilitated by enforcing strict rules on dialog elements and by including explicit links in the dialog code to transition from one dialog element to the next. In contrast to arbitrary VoiceXML code whose execution can be completely non-sequential

due to the presence of "GOTO" tags, a dialog generated by the system 100 has a tree structure, with each path through the tree representing a possible path of dialog execution. This allows the dialog flow to be readily determined and displayed using high level graphical dialog elements, which would be much more difficult with arbitrary VoiceXML.

An application can be saved at any time by selecting a "Save" menu item of the "File" menu of the menubar 410. When an application is saved, the application dialog is translated into an extended VoiceXML format by the dialog transformer 204. Each dialog element in the dialog flow is first translated into corresponding VoiceXML code. Each of the seventeen dialog elements corresponds to one of the seventeen VoiceXML templates 212 that performs the functionality of that element. A VoiceXML template is a sequence of VoiceXML elements that produces the behaviour that the dialog element represents. It is a template because it needs to be configured by the element properties (*e.g.*, name, test condition) which are set by the user, as described above.

Some dialog elements correspond to similar VoiceXML elements (*e.g.*, a Menu dialog element corresponds to a VoiceXML <menu> element), while others map onto a complex sequence of VoiceXML elements (*e.g.*, a Loop dialog element corresponds to multiple VoiceXML <form> elements, each form specifying the next form to execute in an iterative loop). However, even dialog elements that correspond to similar VoiceXML elements represent more functionality than the equivalent VoiceXML element. For example, a Menu dialog element allows prompts to be set by the user, and the Menu dialog element actually maps onto a block of VoiceXML code that contains a <menu> element with embedded <prompt>, <audio>, and other XML elements.

Each dialog element's VoiceXML template is separate from the next and can be sequenced to produce the dialog flow. The sequencing is achieved by a reference at the bottom of each element's template to the next element's template, which causes the templates to be executed in the desired order.

The translation from high-level dialog elements into VoiceXML proceeds as follows. The dialog elements are stored in a tree structure, each branch of the tree corresponding to a path in the dialog flow. The tree is traversed in pre-order traversal to convert each element visited into VoiceXML. For each visited dialog element, VoiceXML code is generated from its corresponding VoiceXML template by filling in the missing or configurable parts of the template using the element properties set by the user, and adding a link to the next element's VoiceXML code at the bottom of the current element's generated VoiceXML code.

Although the forward transformation from dialog flow to VoiceXML is relatively straightforward, the reverse transformation from VoiceXML to dialog flow is more difficult. The sequencing of dialog elements can be recreated from the generated VoiceXML, but property settings for the elements may not be available because some information in the dialog elements is lost when they are converted to VoiceXML. This lost information may not fall within the scope of VoiceXML, and hence, cannot be naturally saved in VoiceXML code. For example, type information for a Form element is used to generate the grammar for that Form. However, the VoiceXML code simply needs to reference the generated Grammar File and is not concerned with the type information itself. Thus, the mapping of the Form element to equivalent VoiceXML code does not include the type information.

To facilitate the reverse translation from VoiceXML code to dialog, the dialog transformer 204 modifies the VoiceXML code by inserting additional attributes into various element tags, providing dialog element information that cannot be stored using the available VoiceXML tags. The resulting file 214 is effectively in an extended VoiceXML format. The additional attributes are stored in a separate, qualified XML namespace so that they do not interfere with the standard VoiceXML elements and attributes, as described in the World Wide Web Consortium's (W3C) *Namespaces in XML* recommendation, available at http://www.w3.org/TR/1999/REC-xml-names-19990114/. This facilitates the parsing of extended VoiceXML files.

Specifically, an extended VoiceXML file can include the following namespace declaration:

```
<vxml version="1.0" xmlns:lq="http://www.telstra.com.au/LyreQuest">
```

This defines a namespace prefix "lq" as bound to the universal resource indicator (URI) http://www.telstra.com.au/LyreQuest. Subsequently, the file may contain the following extended VoiceXML:

```
<form id="SUBROUTINECALL_getMembership"
    lq:element="SubroutineCall"
    lq:name="getMembership"
    lq:calls="sub1.vxml#getMembershipSub">
  <subdialog name="subcall"
      src="sub1.vxml#SUBROUTINE_getMembershipSub">
    <filled>
      <assign name="getMembership.enrich_membership"
          expr="subcall.enrich_membership" lq:element="Output"/>
    </filled>
  </subdialog>
  <block name="link">
    <goto next="#REMOTE_book_flight"/>
  </block>
</form>
```

where the indicated XML tag attributes provide the additional dialog element information, and the remaining code is standard VoiceXML. The additional or extended attributes include the `lq` namespace prefix. The `lq:element`, `lq:name`, and `lq:calls` attributes indicate, respectively, the dialog element that the VoiceXML corresponds to, the name given to that element by the user, and the package and name of the Subroutine element that is being called by the SubroutineCall element. Other elements will have different extended attributes.

The equivalent code in VoiceXML omits the extended attributes, but is otherwise identical:

```
<form id="SUBROUTINECALL_getMembership">
  <subdialog name="subcall"
      src="sub1.vxml#SUBROUTINE_getMembershipSub">
    <filled>
      <assign name="getMembership.enrich_membership"
          expr="subcall.enrich_membership"/>
    </filled>
```

```
        </subdialog>
        <block name="link">
          <goto next="#REMOTE_book_flight"/>
        </block>
      </form>
```

Two extended VoiceXML files, including all the available extended attributes, are listed in Appendix B.

When the application is saved, the dialog transformer 204 also generates a number of other files, including a project file 216, package description files 218, and type description files 220. The project file is given a filename extension of ".lqx", and contains information about the packages (*i.e.*, self-contained groups of files) and other data files making up a project of the voice application development system 100.

An example project file is listed below. Within the project file, the project is defined by a "project" XML element that defines the project name as "mas". Within the "project" element are four sequential "folder" elements that define subdirectories or folders of the directory containing the project file, respectively named Packages, Transcripts, Scenarios, and Generated Code. These folders contain respectively the project's packages, transcripts of text, scenarios of interaction between the corresponding application and a user, and VoiceXML code and grammar generated for one or more specific IVR platforms. Within the "Packages" folder element is a "package" element giving the location and name of any packages used by the project. The "folder" elements can contain one or more "file" elements, each defining the type and name of a file within the encapsulating folder. The "folder" elements can be nested.

```
<?xml version="1.0"?>
<project name="mas">
      <folder name="Packages" directory="packages">
            <package directory="mas" file="mas.pkg.xml"/>
      </folder>
      <folder name="Transcripts" directory="transcripts">
            <file type="transcript" name="mas.in"/>
            <file type="negative" name="mas.negative"/>
      </folder>
      <folder name="Scenarios" directory="scenarios">
```

```
            <file type="scenario" name="mas.scen"/>
            <file type="states" name="mas.states"/>
      </folder>
      <folder name="Generated Code" directory="deploy">
            <folder name="JSGF Code" directory="jsgf">
            </folder>
            <folder name="Nuance Code" directory="nuance">
            </folder>
      </folder>
</project>
```

A package description file is given a filename extension of ".pkg.xml", and contains information about data files belonging to an individual Package of a project. An example of a package description file for the package named "mas" is given below. The file defines the project's dialog file as "mas.vxml", four grammar files, four prompt files, and three type definition files, containing definitions of user-defined variable types. These files are described in more detail below.

```
<?xml version="1.0"?>
<package name="mas">
<file type="dialog" name="mas.vxml"/>
<file type="grammar">
      <file type="rulelist" name="mas.rulelist"/>
      <file type="cover" name="mas.cover"/>
      <file type="slots" name="mas.slots"/>
      <file type="targets" name="mas.targets"/>
</file>
<file type="prompt">
      <file type="rulelist" name="mas.prompts.rulelist"/>
      <file type="cover" name="mas.prompts.cover"/>
      <file type="slots" name="mas.prompts.slots"/>
      <file type="targets" name="mas.prompts.targets"/>
</file>
<file type="typedef" name="cities.type.xml"/>
<file type="typedef" name="fare_class.type.xml"/>
<file type="typedef" name="collection_point.type.xml"/>
</package>
```

A Type description file is given a filename extension of ".type.xml", and contains information about a user-defined Type used in a Package of a project. An example of the file is given below. The file defines an enumerated type named "fare_class" with three possible values: "first", "business", and "economy". The "fare_class" type is associated with four files, respectively defining rules for the grammar, cover (a set of example

phrases), slots (the parameter=value fields that the grammar can return), and targets (more specific slot filling information).

```
<?xml version="1.0" encoding="utf-8"?>
<types>
      <enum name="fare_class">
            <file type="grammar">
                  <file type="rulelist"
            name="fare_class.rulelist"/>
                  <file type="cover" name="fare_class.cover"/>
                  <file type="slots" name="fare_class.slots"/>
                  <file type="targets" name="fare_class.targets"/>
            </file>
            <item name="first"/>
            <item name="business"/>
            <item name="economy"/>
      </enum>
</types>
```

Returning to Figure 3, in order to deploy the application on the IVR 102, the application dialog is translated into VoiceXML by the application builder 206 at step 604. In addition to the dialog, voice applications require grammar and prompts. The application builder 206 generates grammar files 222 and prompts files 224 automatically, using information specified by the user and stored in the dialog, such as prompt wordings and Form input types. This information is supplied by the user entering typical phrases for 'mixed initiative' recognition (*i.e.*, input containing multiple pieces of information). By applying generalisation methods to these phrases, a combinator module of the application development modules 210 generates a starting grammar set capable of handling a large number of input phrases. The application builder 206 also invokes the dialog transformer 204 to create the extended VoiceXML file 214. The grammar 222 and prompts 224 files are used by the IVR code generator 208 to generate VoiceXML 226 for the IVR 102.

A generated grammar file is given a filename extension of ".rulelist". An example of a generated grammar file for a flight booking system is:

```
.Ask_flight_details_destination Cities:X 2 0 destination=$X.cities
.Ask_flight_details_departure_point Cities:X 2 0
departure_point=$X.cities
```

```
.Ask_flight_details_ticket_class Fare_class:X 2 0
ticket_class=$X.fare_class
.Ask_flight_details_date Date:X 2 0 date.day=$X.date.day
date.year=$X.date.year date.day_of_week=$X.date.day_of_week
date.month=$X.date.month date.modifier=$X.date.modifier
.Form_flight_details booking 2 1
.Ask_book_flight_confirmation Confirmation:X 2 0
confirmation=$X.confirmation
.Ask_get_enrich_number_enrich_number Digitstring:X 2 0
enrich_number=$X.digitstring
.Ask_get_collection_point_collection_point Collection_point:X 2 0
collection_point=$X.collection_point
.Ask_another_flight_second_flight Confirmation:X 2 0
second_flight=$X.confirmation
.Ask_get_number_get_number_for_balance Digitstring:X 2 0
get_number_for_balance=$X.digitstring
.Menu_main_menu_booking Booking 2 0
!!Booking booking 3 0
!!Booking 1 0
.Menu_main_menu_membership Membership 2 0
!!Membership membership 3 0
!!Membership 1 0
.Form_flight_details GF_IWantTo:X738 book a Fare_class:X741 class ticket
to Cities:X745 from Cities:X747 on Date:X749 2 1 date.day=$X749.date.day
date.year=$X749.date.year ticket_class=$X741.fare_class
date.day_of_week=$X749.date.day_of_week date.month=$X749.date.month
departure_point=$X747.cities destination=$X745.cities
date.modifier=$X749.date.modifier
.Form_flight_details GF_IWantTo:X750 book a Fare_class:X753 class ticket
to Cities:X757 2 1 ticket_class=$X753.fare_class destination=$X757.cities
```

The first line or rule of this grammar can be used as an example:

```
.Ask_flight_details_destination Cities:X 2 0 destination=$X.cities
```

This grammar rule might be invoked when a flight booking application prompts a customer to provide the destination of a flight. The first field, `.Ask_flight_details_destination`, provides the name of the grammar rule. The second field, `Cities:X`, indicates that the customer's response `X` is of type Cities. This type is defined by its own grammar that includes a list of available city names. The following two fields, `2 0`, are used for grammar learning, as described in International Patent Publication No. WO 00/78022, *A Method of Developing an Interactive System*. The first field indicates the number of training examples that use the grammar rule. The second field indicates the number of other rules that refer to the rule. The last field, `destination=$X.cities`, indicates that the result of the rule is that the parameter

destination is assigned a value of type Cities having the value of x. A more complex example is provided by the last rule :

```
.Form_flight_details GF_IWantTo:X750 book a Fare_class:X753 class ticket
to Cities:X757 2 1 ticket_class=$X753.fare_class destination=$X757.cities
```

In this case, the grammar rule invokes three other grammars: GF_IwantTo, Fare_class, and Cities and assigns the results to parameters named X750, X753, and X757, respectively. This rule defines the application parameters ticket_class and destination.

A prompts file is given a filename extension of ".prompts.rulelist", and each line of the file defines the speech prompt that is to be provided to a user of the application when the corresponding element of the dialog is executed. An example of a generated prompts file is:

```
.Goodbye thank you for using this application. goodbye. 1 1
.Noinput sorry, i did not hear anything. 1 1
.Nomatch sorry, i did not understand you. 1 1
.Ask_flight_details_destination where would you like to go 1 1
.Ask_flight_details_destination_2 please say the destination city name 1
1
.Help_Ask_flight_details_destination please say the destination.  1 1
.Ask_flight_details_departure_point where would you like to fly from 1 1
.Ask_flight_details_departure_point_2 please say the departure point 1 1
.Help_Ask_flight_details_departure_point please say the departure point.
1 1
.Ask_flight_details_ticket_class what class would you like to fly 1 1
```

The format of the prompts file is the same as the grammar file. This allows the prompts to be improved through machine learning as though they were a grammar, using a grammar learning method such as that described in International Patent Publication No. WO 00/78022, *A Method of Developing an Interactive System*.

The generated prompts include dynamically prompts. An example of a dynamic prompt is: "You have selected to buy Telstra shares. How many of the Telstra shares would you like to buy?". The word, "Telstra" is dynamically inserted into the application's prompt to the user.

The voice application development system 100 generates text-to-speech (TTS) prompts within the VoiceXML code that are evaluated on the fly. Although VoiceXML syntax allows an expression to be evaluated and played as a TTS prompt, the system 100 extends this by allowing an ECMAscript or JavaScript function to be called to evaluate each variable used in a prompt. By evaluating variables in a function rather than as an inline expression, complex test conditions can be used to determine the most suitable prompt given the available information in the variables. This might result in a prompt, for example, of "six dollars" rather than "six dollars and zero cents". In addition to automatically generating and incorporate JavaScript function calls in VoiceXML, the system 100 also generates the corresponding JavaScript functions by incorporating user-supplied prompt text and variables into the JavaScript templates 230. This allows the user to develop a voice application with dynamically generated prompts without having to manually code any JavaScript.

For example, an automatically generated function call for a prompt named PromptConfirm_payment_details is:

```
<field name="confirm">
      <grammar src="mas.gsl#Confirm"/>
      <prompt>
            <value expr="PromptConfirm_payment_details(
                  payment_details.company,
                  payment_details.amount,
                  payment_details.payment_date)"/>
      </prompt>
```

The corresponding JavaScript prompt function generated by the system 100 is:

```
function PromptConfirm_payment_details(company,amount,payment_date)
{
    var result;
    result = "the company is " + Bpay_names(company)  +
             "the amount is " + Money(amount)  +
             "the payment date is " + Date(payment_date)  +
             "is this correct? ";
    if( valid_string(result))
    {
          return result;
```

```
        }
    ;
        return result;
    }
```

The system 100 represents prompts using a language model that describes all of the prompts that can be played, along with their meanings. This model contains the same type of information as a speech recognition grammar, and therefore the prompts to be played can be represented using a grammar. Prompts to be generated by the application are first represented as a grammar to enable that grammar to be improved using techniques such as grammar learning, as described in International Patent Publication No. WO 00/78022, *A Method of Developing an Interactive System*. The grammar is subsequently converted into JavaScript and referenced by the application's VoiceXML tags, as described above.

An example of a prompt represented as a grammar is:

```
.Confirm_payment_details the company is Bpay_names:x1 the amount is
Money:x2 the payment date is Date:x3 is this correct? 1
company=$x1.company amount.dollars=$x2.amount.dollars
amount.cents=$x2.amount.cents payment_date=$x3.payment_date
```

Returning to Figure 3, after the application has been built at step 604, that is, the extended VoiceXML 214, the grammar 222 and the prompts 224 for the application have been generated by the application builder 206, the application can be tested and further developed at steps 606 to 610. Steps 606, 608, 610 and 612 can be executed in any order. At step 606, the application can be simulated and refined. This involves simulating the execution of the application and refining its accuracy by allowing the user to tag phrases that do not match the existing grammar. When user input during testing does not match the grammar, a dialog box is displayed, allowing the user to tag the phrase and supply the correct 'slot' or slots corresponding to that input. A slot is a parameter=value pair for the application, such as "fare_class=business". A grammar learner module of the application development modules 210 then uses a transcript of the simulation to update the application grammar 222. New phrases learnt from this grammar are then displayed to the user, who can manually tag individual phrases as being incorrect. At step 608, the grammar

capabilities of the application can be further improved by a Train Grammar tool of the application development modules 210. This is similar to the simulate and refine module, but allows the user to enter a list of typical user responses for each application prompt. At step 610, a Generate Scenarios module of the application development modules 210 generates scenarios of possible interactions between the application and a human. Based on these scenarios, the user can determine whether and application prompts need improvement.

When the application has been tested and is ready for use, the IVR code generator 208 executes a code generation process at step 612 to generate pure VoiceXML suitable for a particular speech-enabled IVR such as the IVR 102 of Figure 1. As shown in Figure 6, the code generation process begins at step 702 by removing the extended attributes from the extended VoiceXML file 214 to generate pure VoiceXML. At step 704, prompts (including dynamic prompts) in the prompt file 224 are converted into JavaScript functions. At step 706, these JavaScript functions are incorporated into the pure VoiceXML by adding references to the functions in VoiceXML tags, and adding the functions themselves to the pure VoiceXML. At step 708, the IVR grammar file 228 is generated by translating the application grammar file 222 into a grammar format supported by the desired IVR platform, such as Nuance™ GSL or generic VoiceXML 1.0 JSGF grammar format, as selected by the user. Other grammar formats can be supported in the same way. At step 710, references to the IVR grammar file 228 are incorporated into the pure VoiceXML. The result is the pure VoiceXML file 226. The VoiceXML file 226 and the IVR grammar file 228 are sufficient to deploy the voice application on the IVR 102.

For the purposes of illustration, Appendix C provides a partial listing of a pure VoiceXML file corresponding to the first extended VoiceXML file listed in Appendix B. The listing in Appendix C includes the VoiceXML with the merged JavaScript for supporting Prompts. The JavaScript code is at the end of the listing.

Many modifications will be apparent to those skilled in the art without departing from the scope of the present invention as herein described with reference to the accompanying drawings.

APPENDIX A: DIALOG ELEMENTS

Decision	
Description:	A point in the dialog where a decision is made to determine which path of execution should continue.
Notes:	Each Decision element is followed by one or more Decision-Branch elements.
Parameters:	**Decision** **Name:** a name for this Decision element

Decision branch	
Description:	A path that is followed if the condition is true.
Notes:	Can only be added after a Decision element.
Parameters:	**Branch** **Condition:** an ECMAScript expression that if evaluated to true will cause execution to continue down this branch, *eg. x == 2, validity == true, currency == 'AUD'*. Note that strings should be quoted in single quotes. An "else" condition is created by enabling the Otherwise checkbox.

Menu	
DESCRIPTION:	Prompts the user to select from a series of menu choices
Notes:	Each Menu element is followed by one or more Menu-Choice. Prompts for the menu can include: (1) Auto generated prompts (by default); (2) User specified prompts; (3) Audio file for playing prompts. In addition, tapered prompts are supported.
Parameters:	**Menu** **Name:** name for this Menu element. **Prompts** **Tapered prompts:** the tapered prompts played to ask for a menu choice. The system can generate a default text prompt based on the menu choices and a default audio file name for recording the corresponding audio prompt, or you may enter in your own values. If an audio prompt is not required, the "Generate default audio file name" checkbox should be unchecked and the "Audio file" text box should be left empty. **Help prompt:** the prompt played when the user requests help. As with tapered prompts, the system can generate a default text prompt and audio file name, or you may enter in your own.

5

Menu choice	
Description:	Defines a specific menu choice
Notes:	Can only be added after a Menu element. This is the only Element whose name can have a space.
Parameters:	**Choice** **Choice:** a word or phrase for this menu choice.

Form	
Description:	Collects user inputs to fill in a 'form'. Defines a type and prompts for each input. These can be standard types such as Time and Date, or user defined such as a list of products.
Notes:	User-defined-types need to be first created in the LHS Dialog/Project Files window, by right-clicking on 'package' icon. Selecting "Hot link" allows jumping to the Form from anywhere in the dialog. The Forms support mixed initiative filling of the Fields. Other Elements can refer to Form Fields using: "FormName.FieldName". When entering Form Fields, default string values need to be in single quotes (eg. 'telstra').
Parameters:	**Form** **Name:** a name for this Form element. An object with this name will be created to store the form fields. Each field can then be accessed as $formName.$fieldName, *eg. BuyForm.price*. **Modal:** if enabled, then only that input field's grammar is enabled when collecting each field; all other grammars are temporarily disabled (including hot-linked global grammars). This means that you cannot get out of this form by saying a hotword. For example, a form that collects a login pin should be made modal. **Confirm All:** enable confirmation of all form fields together at the end. **Hot link:** enable jumping to the form from anywhere in the document, by creating a document grammar for the form. **Input Field** **Name:** name of slot to be filled. **Type:** type of slot value. **Default:** default value for slot. If a default is given, the user will not be prompted for the field. If the field type is a structured object, the slot will have multiple properties, one for each of the object's properties. You may specify a default by filling in some or all of these properties, depending on what is valid for a particular type. Note that strings should be quoted in single quotes, *eg. 'wednesday'*. **Confirm:** enable individual confirmation of this field. This is in addition to the form confirmation of all fields at the end. **Input Field Prompts** **Tapered prompts:** the tapered prompts played to collect input for this field. The system can

	generate a default text prompt based on the field name and a default audio file name for recording the corresponding audio prompt, or you may enter in your own values. If an audio prompt is not required, the "Generate default audio file name" checkbox should be unchecked and the "Audio file" text box should be left empty. **Help prompt:** the prompt played when the user requests help. As with tapered prompts, the system can generate a default text prompt and audio file name, or you may enter in your own.

Record	
Description:	Captures a voice recording from the user
Notes:	Note that the Record duration is in seconds. Setting it to too high will result in excesive memory usage & possible harddisk limit problems.
PARAMETERS:	**Record** **Name:** a name for this Record element. **MIME-type:** MIME-type storage format for the recording. This field may be left blank if the IVR does not allow a MIME-type specification. Please consult your IVR documentation for supported MIME-types. **Beep:** enable a beep to be sounded before recording starts. **DTMF Terminate:** enable a DTMF key press to terminate the recording. **Duration:** maximum recording time in seconds. **Confirm:** enable confirmation of the recorded material. **Prompt:** the prompt played to ask for a recording. The system can generate a default text prompt and a default audio file name for recording the corresponding audio prompt, or you may enter in your own values. If an audio prompt is not required, the "Generate default audio file name" checkbox should be unchecked and the "Audio file" text box should be left empty.

Speaker	
Description:	Plays TTS & audio to the caller.
Notes:	Can say variables of predefined types, such as Date, Money, etc. The variables need to be declared as a valid object (unless it is created via a Form Field). Eg. User can use local processing to declare a variable called price, as follows: price = new Object(); price.dollars = 10; price.cents = 4; To use it in Speaker, type the following: PromptMoney(price) in a Speaker expression fragment. To play back predefined Types, such as Money: (1) add a Speaker fragment of the type "Expression"; (2) enter in the "Expression" text box: PromptType(x), where Type is the predefined type name, x is the variable of the predefined type. E.g. If "y" is type money (where y.dollars=5 y.cents=0), then entering PromptMoney(y) will result in the following being played: "five dollars".
Parameters:	**Speaker**

	Name: a name for this Speaker element. **Fragment** **Type:** type of prompt fragment. **Text:** a fixed text fragment. The text specifies text-to-speech (TTS) that should be played. The audio file URL specifies a recording that should be played, in precedence to the text-to-speech, *eg. greeting.wav, http://server/welcome.wav*. Either of the audio file or the text may be left empty to request that only TTS or audio with no alternate text should be played. **Expression:** an evaluated prompt fragment. The ECMAScript expression is evaluated and played as text-to-speech (TTS), eg. recordedMessage, counter + 1. To play back a variable of a known type, you should use the function PromptType(x), where Type is the name of that type, and x is the name of the variable, eg. PromptMoney(BuyForm.price). This is particularly important for playing back structured types. The audio expression is evaluated and retrieved as an audio file URL for playing, eg. company + '.wav'. The expressions may reference any variable defined in the package, such as in a Variables element or a Form object. Either of the audio expression or the TTS expression may be left empty to request that only TTS or audio with no alternate text should be played.

Local Processing	
Description:	Local computation.
Notes:	This can be any ECMAScript code, eg. functions, statements, function calls. Any variable declared in the dialog may be accessed, eg. declared Variables, Form fields, SubroutineCall outputs, RemoteProcessing outputs.
Parameters:	**Local Processing** **Name:** a name for this Local Processing element. **ECMAScript:** any arbitrary ECMAScript code, such as variable declarations, if blocks, assignment statements, function declarations and function calls, *eg. x = x + 1; currency = 'AUD';*. Note that strings should be quoted in single quotes.

Remote Processing	
Description:	Call a server-side script (via a HTTP URL) to perform some processing. The script should return a VoiceXML document with the results of the processing.
NOTES:	Need PHP/CGI running on remote Web server to handle this. Name of the input and output parameters needn't be declared in the Variables element. The names of the input and output parameters should match what is required at the remote server end. Other Dialog Elements can refer to the output of the Remote Processing element using: "RemoteProcName.outputName". See Chapter 9: Advanced Techniques for more information
Parameters:	**Remote Processing** **Name:** a name for this Remote Processing element. An object with this name will be created to store the returned outputs. Each output can then be accessed as $remoteProcessingName.$output, *eg. ValidatePin.validity* **Source URL:** the URL of the remote script to execute, *eg. http://server/cgi-bin/script1*.

	Input **Name:** name of the input parameter to be submitted, *eg. price*. This is the name that the server script expects. **Value:** value of the input parameter to be submitted, *eg. BuyForm.price, 1, 'AUD', true, x + 1*. This may be any valid ECMAScript expression. The expression may reference any variable defined in the package, such as in a Variables element or a Form object. If the value is an ECMAScript object with fields f1, f2, ..., then the object is serialised and submitted using the names o.f1, o.f2, ..., assuming o is the declared input name. **Output** **Name:** name of each parameter returned by the remote script, *eg. outField1*. The remote script should return a VoiceXML document containing a subdialog form whose return namelist includes all these output fields.

Loop Call	
Description:	Calls a Loop to execute a part of a dialog that should be iterated several times.
Notes:	A corresponding Loop element is required. See Chapter 9: Advanced Techniques for more information
Parameters:	**Loop Call** **Name:** a name for this Loop Call element. **Source:** the loop to call. The loop must be defined in this package.

Subroutine Call	
Description:	Start a subroutine. Used for breaking bigger programs into smaller components, for ease of readability, reuse of code, and support of pre-packaged code.
Notes:	A corresponding Subroutine element is required. Input parameter names needn't be declared. Ouput parameter names are set by the Subroutine element. Other Dialog Elements can refer to the output parameters by calling: "SubroutineCallName.OutputParamName". See Chapter 9: Advanced Techniques for more information
Parameters:	**Subroutine Call** **Name:** a name for this Subroutine Call element. An object with this name will be created to store the returned outputs. Each output can then be accessed as $subName.$output, *eg. ValidatePin.validity*. **Source:** the subroutine to call (qualified with the package name). If the declared subroutine inputs and outputs have changed, you may need to reselect the subroutine from the Source list box to refresh the displayed inputs and outputs. **Inputs**

	This is a list of the inputs that the called subroutine expects. For each input, you must specify an ECMAScript expression whose value will be passed to the subroutine, *eg. 0, BuyForm.price, pinValidity, 'AUD'*. It may reference any variable defined in the package, such as in a Variables element or a Form object. Note that strings should be quoted in single quotes. Note also that you cannot leave any input value blank. **Outputs** This is a list of the outputs that the called subroutine returns.

Jump	
Description:	Jump to a predefined block. Can jump to any element in the same MainDialog or Subroutine that has a name.
Notes:	Valid destinations for Jump: (1) Jumps within a MainDialog; (2) Jump from a Loop in a MainPackage to the MainDialog; (3) Within a subroutine.
Parameters:	**Jump** **Destination:** the destination element to jump to. You can only jump within a main dialog or within a subroutine. You cannot jump to an element in a loop or in another package. The available elements to jump to are presented in the drop-down list box.

End	
Description:	Terminate the call. End of dialog
Notes:	
Parameters:	Causes execution to terminate immediately

Transfer	
Description:	Transfer the call to another number.
Notes:	
Parameters:	**Transfer** **Name:** a name for this Transfer element. **Destination:** a number to dial or an ECMAScript expression that evaluates to such a number. A valid number is a string of digits with optional spaces, *eg. 1234567, 03 1234567*. The number may optionally contain a protocol specifier. Please consult your IVR documentation for specific supported number formats. **Connection Timeout:** maximum time in seconds to wait for a connection before a failure is reported.

Loop Break	
Description:	Used within a Loop to break out of the loop.
Notes:	Only valid within a Loop
Parameters:	Causes execution to break out of the loop immediately. The loop exit message, if any, is not

	played

Loop Next	
Description:	Used within a Loop to indicate the end of one iteration.
Notes:	Only valid within a Loop
Parameters:	Causes execution of the next iteration of the loop, if the loop test condition evaluates to true. The loop step is executed before the condition is evaluated.

Subroutine Return	
Description:	Return from subroutine
Notes:	Only valid within a Subroutine
Parameters:	Return from a subroutine call.

Variables	
Description:	Declare "global" variables that can be accessed from anywhere within this Package, eg. inside a Loop, LocalProcessing, Speaker element.
Notes:	Form Fields, Subroutine Call outputs and Remote Processing outputs do not need to be declared. They are automatically created.
Parameters:	**Variables** **Name:** a name for the variable. The name should be unique within this package. **Value:** an ECMAScript expression, usually a constant, that sets the initial value for the variable, eg. 0, 'cat', true . Note that strings should be quoted with single quotes. The value may be left empty to create an ECMAScript undefined value.

Hotwords	
Description:	Create "hot words" that transition the user to a specified point in the dialog when they are uttered by the user.
Notes:	Allows the user to jump from one part of the dialog to another if they want to do something else, for example. Hotwords can only be created in the Main Package, not in Subroutine Packages.
Parameters:	**Hotwords** **Hotword:** a word or phrase that will trigger a jump. **Destination:** the destination element to jump to.

5

Start Dialog	
Description:	The entry point for the application.
Notes:	This is automatically created when you create a new project. Only the first package has a Start Dialog element. Each project must have one and only one Start Dialog element. You cannot add more Start Dialogs, nor delete any.

Parameters:	**Start Dialog** This is the entry point for the application. **Name:** a name for this Start Dialog element. **Quit Prompt:** the prompt played when the user has requested to quit, after which the dialog will terminate. The system can generate a default text prompt and a default audio file name for recording the corresponding audio prompt, or you may enter in your own values. If an audio prompt is not required, the "Generate default audio file name" checkbox should be unchecked and the "Audio file" text box should be left empty. **No Input Prompt:** the prompt played when no input is detected while the user is being asked for a response. The user is then reprompted for a response. As with the Quit prompt, the system can generate a default text prompt and a default audio file name, or you may enter in your own values. **No Match Prompt:** the prompt played when the detected user response cannot be recognised by the system. The user is then reprompted for another response. As with the Quit prompt, the system can generate a default text prompt and a default audio file name, or you may enter in your own values.

Subroutine	
Description:	This is the entry point for a subroutine.
Notes:	Subroutine elements cannot be added to the Main Package. They can only be added to non-Main Packages. More than one Subroutine can be added to a non-Main Package. However, all Subroutine inputs and outputs must be unique within the Package, ie. no two Subroutines can declare the same input name or output name. Furthermore, no input name can be the same as an output name. The Subroutine element is created by (1) right-clicking on the "Project" icon on the Dialogs/Project Files window (LHS) to add a new Package; and (2) right-clicking on a Package icon on the Dialogs window (LHS) to add a new Subroutine. Each path in the Subroutine should end with a Return element, otherwise the Subroutine will not return to the calling dialog. See Chapter 9: Advanced Techniques for more information
Parameters:	**Subroutine** **Name:** a name for this Subroutine element. **Input** **Name:** name of each input parameter expected, eg. pin. The input will be declared as a document variable that you can access from anywhere within the package. You do not need to (re)declare it under the Variables element. **Output**

	Name: name of each return parameter, eg. validity. The output will be declared as a document variable that you can access from anywhere within the package. You do not need to (re)declare it under the Variables element.

Loop	
Description:	A portion of the dialog that should be executed several times.
Notes:	The Loop element is created by right-clicking on the Package icon on the Dialogs window (LHS). Variables are freely shared between the Loop body and the main dialog (of the same Package) as they are in one VoiceXML document. See Chapter 9: Advanced Techniques for more information
Parameters:	**Loop** **Name:** a name for this Loop element. **Loop Test:** an ECMAScript expression that if evaluated to true will cause execution of the next iteration of the loop, eg. counter < 5. **Test at start:** If enabled, the test condition is evaluated at the start of the loop and the loop is equivalent to a while / for loop. If disabled, the loop body is executed before the test condition is evaluated and the loop is equivalent to a do-while loop. **Exit message:** a message to be played when the loop exits normally, eg. there are no more items. The system can generate a default text prompt and a default audio file name for recording the corresponding audio prompt, or you may enter in your own values. If a message is not required, the "Generate default..." checkboxes should be unchecked and the "Audio file" and "TTS" text boxes should be left empty. **Loop Init** Variables to be initialised at the start of the loop. **Name:** name of a variable, eg. counter. The variable must have been created elsewhere, such as in the Variables element. **Value:** an ECMAScript expression that sets the initial value for the variable, eg. 0. **Loop Step** Variables to be incremented before another iteration of the loop. The increment occurs before the test condition is reevaluated. **Name:** name of a variable, eg. counter. **Value:** an ECMAScript expression to increment the variable, eg. counter + 1.

APPENDIX B: TWO EXTENDED VoiceXML files OF AN APPLICATION

FILE 1. DEMO2_8MAY.VXML

```
<?xml version="1.0" encoding="utf-8"?>
<vxml version="1.0" xmlns:lq="http://www.telstra.com.au/LyreQuest">
      <link event="help">
            <grammar src="null#Help"/>
      </link>

      <link event="repeat">
            <grammar src="null#Repeat"/>
      </link>

      <link event="quit">
            <grammar src="null#Quit"/>
      </link>

      <script>
            function merge(obj1, obj2)
            {
                  if (obj1 == undefined)
                        return obj2;
                  else
                        return obj1;
            }
      </script>

      <var name="_dummy"/>

      <var name="counter" lq:element="DocumentVariables"/>

      <catch event="quit">
            <prompt count="1" bargein="true" lq:text="">
                  <value expr="PromptGoodbye()"/>
            </prompt>
            <exit/>
      </catch>

      <catch event="noinput">
            <prompt count="1" bargein="true" lq:text="">
                  <value expr="PromptNoinput()"/>
            </prompt>
            <reprompt/>
      </catch>

      <catch event="nomatch">
            <prompt count="1" bargein="true" lq:text="">
                  <value expr="PromptNomatch()"/>
            </prompt>
            <reprompt/>
      </catch>

      <form id="DIALOG_MainDialog" lq:element="StartDialog"
lq:name="MainDialog">
```

```
                <block name="link">
                        <goto next="#SPEAKER_sayWelcome"/>
                </block>

        </form>

        <form id="SPEAKER_sayWelcome" lq:element="Speaker"
lq:name="sayWelcome">
                <block>
                        <prompt>
                                Welcome to the automated X Y Z bank
                        </prompt>
                </block>
                <block name="link">
                        <goto next="#LOOPCALL_call_authenticate_loop"/>
                </block>

        </form>

        <form id="LOOPCALL_call_authenticate_loop" lq:element="LoopCall"
lq:name="call_authenticate_loop" lq:calls="authenticate">
                <subdialog name="loopcall" src="#LOOP_authenticate">
                        <param name="counter" expr="document.counter"/>
                        <param name="get_PIN" expr="document.get_PIN"/>
                        <param name="validatePIN" expr="document.validatePIN"/>
                        <param name="call_get_balance"
expr="document.call_get_balance"/>
                        <param name="payment_details"
expr="document.payment_details"/>
                        <filled>
                                <assign name="document.counter"
expr="loopcall.counter"/>
                                <assign name="document.get_PIN"
expr="loopcall.get_PIN"/>
                                <assign name="document.validatePIN"
expr="loopcall.validatePIN"/>
                                <assign name="document.call_get_balance"
expr="loopcall.call_get_balance"/>
                                <assign name="document.payment_details"
expr="loopcall.payment_details"/>
                        </filled>
                </subdialog>
                <block name="link">
                        <goto next="#DECISION_checkPINvalidity"/>
                </block>

        </form>

        <form id="DECISION_checkPINvalidity" lq:element="Decision"
lq:name="checkPINvalidity">
                <block>
                        <if cond="validatePIN.x == &apos;PASS&apos;">
                                <goto next="#BRANCH_checkPINvalidity_0"/>
                        <else/>
                                <goto next="#BRANCH_checkPINvalidity_1"/>
                        </if>
                </block>
```

```
        </form>

        <form id="BRANCH_checkPINvalidity_0" lq:element="Branch"
  lq:condition="validatePIN.x == &apos;PASS&apos;">
              <block name="link">
                    <goto next="#MENU_main"/>
              </block>

        </form>

        <menu id="MENU_main" lq:element="Menu" lq:name="main">
              <prompt count="1" bargein="true" lq:text="">
                    <value expr="PromptMenu_main()"/>
              </prompt>
              <choice next="#MENUITEM_main_balance">
                    <grammar src="null#Menu_main_balance"/>
              </choice>
              <choice next="#MENUITEM_main_enquiry">
                    <grammar src="null#Menu_main_enquiry"/>
              </choice>
              <choice next="#MENUITEM_main_leave_message">
                    <grammar src="null#Menu_main_leave_message"/>
              </choice>
              <choice next="#MENUITEM_main_pay_bills">
                    <grammar src="null#Menu_main_pay_bills"/>
              </choice>
              <choice next="#MENUITEM_main_quit">
                    <grammar src="null#Menu_main_quit"/>
              </choice>

              <catch event="help">
                    <prompt count="1" bargein="true" lq:text="">
                          <value expr="PromptHelp_Menu_main()"/>
                    </prompt>
              </catch>
              <catch event="repeat">
                    <reprompt/>
              </catch>
        </menu>

        <form id="MENUITEM_main_balance" lq:element="MenuChoice"
  lq:name="balance">
              <block name="link">
                    <goto next="#SUBROUTINECALL_call_get_balance"/>
              </block>

        </form>

        <var name="call_get_balance" expr="new Object()"
  lq:references="call_get_balance"/>
        <form id="SUBROUTINECALL_call_get_balance"
  lq:element="SubroutineCall" lq:name="call_get_balance"
  lq:calls="subroutine1.vxml#sub_present_balance">
              <subdialog name="subcall"
  src="subroutine1.vxml#SUBROUTINE_sub_present_balance">
                    <param name="pin_number" expr="get_PIN.PIN"
  lq:element="Input"/>
```

```
                        <filled>
                              <assign name="call_get_balance.result_code"
    expr="subcall.result_code" lq:element="Output"/>
                        </filled>
                  </subdialog>
                  <block name="link">
                        <goto next="#SPEAKER_Speaker24"/>
                  </block>

            </form>

            <form id="SPEAKER_Speaker24" lq:element="Speaker"
    lq:name="Speaker24">
                  <block>
                        <prompt>
                              the result of your balance enquiry is
                              <value expr="call_get_balance.result_code"/>
                        </prompt>
                  </block>
                  <block lq:element="Goto" lq:next="main">
                        <goto next="#MENU_main"/>
                  </block>

            </form>

            <form id="MENUITEM_main_enquiry" lq:element="MenuChoice"
    lq:name="enquiry">
                  <block name="link">
                        <goto next="#SPEAKER_Speaker21"/>
                  </block>

            </form>

            <form id="SPEAKER_Speaker21" lq:element="Speaker"
    lq:name="Speaker21">
                  <block>
                        <prompt>
                              you are being transfered to an operator
                        </prompt>
                  </block>
                  <block name="link">
                        <goto next="#TRANSFER_Transfer22"/>
                  </block>

            </form>

            <form id="TRANSFER_Transfer22" lq:element="Transfer"
    lq:name="Transfer22" lq:bridge="false" lq:destination="0392536742"
    lq:destinationExpression="null" lq:connectTimeout="30">
                  <transfer name="Transfer22" bridge="false"
    connecttimeout="30s" dest="0392536742">
                  </transfer>
            </form>

            <form id="MENUITEM_main_leave_message" lq:element="MenuChoice"
    lq:name="leave message">
                  <block name="link">
```

```
                <goto next="#RECORD_Record20"/>
            </block>

        </form>

        <var name="Record20" lq:references="Record20"/>
        <form id="RECORD_Record20" lq:element="Record" lq:name="Record20"
lq:beep="true" lq:duration="30" lq:dtmfTerminate="true"
lq:mimeType="null" lq:confirm="true">
            <record name="record_Record20" beep="true" maxtime="30s"
dtmfterm="true">
                <prompt count="1" bargein="true" lq:text="at the tone,
please record your message. you can press hash to end.">
                    <value expr="PromptRecord_Record20()"/>
                </prompt>
            </record>

            <field name="confirm" modal="true">
                <grammar src="null#Confirm"/>
                <prompt>
                    Your recording is <value expr="record_Record20"/>
To keep it, say yes. To discard it, say no.
                </prompt>
                <filled>
                    <if cond="confirm != 'yes'">
                        <clear/>
                    </if>
                </filled>
                <catch event="help">
                    <prompt>
                        Please say yes to keep your recording, or
say no to record another message.
                    </prompt>
                </catch>
                <catch event="repeat">
                    <prompt>
                        Your recording is <value
expr="record_Record20"/> To keep it, say yes. To discard it, say no.
                    </prompt>
                </catch>
            </field>

            <block>
                <assign name="Record20" expr="record_Record20"/>
            </block>

            <block lq:element="Goto" lq:next="main">
                <goto next="#MENU_main"/>
            </block>

        </form>

        <form id="MENUITEM_main_pay_bills" lq:element="MenuChoice"
lq:name="pay bills">
            <block name="link">
                <goto next="#FORM_payment_details"/>
            </block>
```

```
        </form>

        <var name="payment_details" expr="new Object()"
lq:references="payment_details"/>
        <form id="FORM_payment_details" lq:element="Form"
lq:name="payment_details" lq:modal="false" lq:confirm="true"
lq:documentGrammar="true">
            <grammar scope="document" src="null#Form_payment_details"/>

            <field name="company" cond="false"/>
            <field name="amount" cond="false"/>
            <field name="payment_date" cond="false"/>
            <block>
                <assign name="payment_details.company" expr="company"/>
                <assign name="payment_details.amount" expr="amount"/>
                <assign name="payment_details.payment_date"
expr="payment_date"/>
            </block>
            <subdialog name="payment_details_company"
src="#ASK_payment_details_company">
                <param name="payment_details" expr="payment_details"/>
                <filled>
                    <assign name="payment_details"
expr="payment_details_company.payment_details"/>
                </filled>
            </subdialog>

            <subdialog name="payment_details_amount"
src="#ASK_payment_details_amount">
                <param name="payment_details" expr="payment_details"/>
                <filled>
                    <assign name="payment_details"
expr="payment_details_amount.payment_details"/>
                </filled>
            </subdialog>

            <block>
                <if cond="payment_details.payment_date == undefined">
                    <script>
                        <![CDATA[
                            payment_details.payment_date = new
Object();
                            payment_details.payment_date.modifier
= 'today';
                        ]]>
                    </script>
                </if>
            </block>

            <subdialog name="payment_details_payment_date"
src="#ASK_payment_details_payment_date">
                <param name="payment_details" expr="payment_details"/>
                <filled>
                    <assign name="payment_details"
expr="payment_details_payment_date.payment_details"/>
                </filled>
```

```
                </subdialog>

                <field name="confirm">
                      <grammar src="null#Confirm"/>
                      <prompt>
                            <value
    expr="PromptConfirm_payment_details(payment_details.company,
    payment_details.amount, payment_details.payment_date)"/>
                      </prompt>
                      <filled>
                            <if cond="confirm != 'yes'">
                                  <assign name="payment_details.company"
    expr="undefined"/>
                                  <assign name="payment_details.amount"
    expr="undefined"/>
                                  <assign name="payment_details.payment_date"
    expr="undefined"/>
                                  <clear/>
                            </if>
                      </filled>
                      <catch event="help">
                            <prompt>
                                  <value
    expr="PromptHelp_Confirm_payment_details()"/>
                            </prompt>
                      </catch>
                      <catch event="repeat">
                            <reprompt/>
                      </catch>

                </field>
                <block lq:element="Goto" lq:next="main">
                      <goto next="#MENU_main"/>
                </block>

          </form>

          <form id="ASK_payment_details_company" lq:element="FormField"
    lq:references="payment_details" lq:name="company" lq:type="bpay_names"
    lq:default="" lq:confirm="false">
                <grammar src="null#Ask_payment_details_company"/>

                <var name="payment_details"/>

                <block>
                      <if cond="payment_details.company != undefined">
                            <return namelist="payment_details"/>
                      </if>
                </block>

                <field name="company" cond="false"/>
                <field name="amount" cond="false"/>
                <field name="payment_date" cond="false"/>
                <initial name="start">
                      <prompt count="1" bargein="true" lq:text="please say
    the company">
```

```
                        <value
    expr="PromptAsk_payment_details_company()"/>
                    </prompt>
                </initial>
                <filled mode="any">
                    <assign name="payment_details.company"
    expr="merge(payment_details.company, company)"/>
                    <assign name="payment_details.amount"
    expr="merge(payment_details.amount, amount)"/>
                    <assign name="payment_details.payment_date"
    expr="merge(payment_details.payment_date, payment_date)"/>
                    <if cond="payment_details.company == undefined">
                        <clear/>
                    </if>
                </filled>

                <block>
                    <return namelist="payment_details"/>
                </block>


                <catch event="help">
                    <prompt count="1" bargein="true" lq:text="please say
    the company. ">
                        <value
    expr="PromptHelp_Ask_payment_details_company()"/>
                    </prompt>
                </catch>

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


            </form>

            <form id="ASK_payment_details_amount" lq:element="FormField"
    lq:references="payment_details" lq:name="amount" lq:type="Money"
    lq:default="[]" lq:confirm="false">
                <grammar src="null#Ask_payment_details_amount"/>

                <var name="payment_details"/>

                <block>
                    <if cond="payment_details.amount != undefined">
                        <return namelist="payment_details"/>
                    </if>
                </block>

                <field name="amount" cond="false"/>
                <field name="payment_date" cond="false"/>
                <initial name="start">
                    <prompt count="1" bargein="true" lq:text="">
                        <value
    expr="PromptAsk_payment_details_amount()"/>
                    </prompt>
                </initial>
                <filled mode="any">
```

```
                        <assign name="payment_details.amount"
    expr="merge(payment_details.amount, amount)"/>
                        <assign name="payment_details.payment_date"
    expr="merge(payment_details.payment_date, payment_date)"/>
                        <if cond="payment_details.amount == undefined">
                              <clear/>
                        </if>
                  </filled>

                  <block>
                        <return namelist="payment_details"/>
                  </block>


                  <catch event="help">
                        <prompt count="1" bargein="true" lq:text="">
                              <value
    expr="PromptHelp_Ask_payment_details_amount()"/>
                        </prompt>
                  </catch>

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

            </form>

            <form id="ASK_payment_details_payment_date" lq:element="FormField"
      lq:references="payment_details" lq:name="payment_date" lq:type="Date"
      lq:default="[{modifier='today'}]" lq:confirm="true">
                  <grammar src="null#Ask_payment_details_payment_date"/>

                  <var name="payment_details"/>

                  <block>
                        <if cond="payment_details.payment_date != undefined">
                              <assign name="start" expr="true"/>
                              <goto nextitem="confirm"/>
                        </if>
                  </block>

                  <field name="payment_date" cond="false"/>
                  <initial name="start">
                        <prompt count="1" bargein="true" lq:text="">
                              <value
    expr="PromptAsk_payment_details_payment_date()"/>
                        </prompt>
                  </initial>
                  <filled mode="any">
                        <assign name="payment_details.payment_date"
    expr="merge(payment_details.payment_date, payment_date)"/>
                        <if cond="payment_details.payment_date == undefined">
                              <clear/>
                        </if>
                  </filled>

                  <field name="confirm">
```

```
                        <grammar src="null#Confirm"/>
                        <prompt>
                              <value
    expr="PromptConfirm_payment_details_payment_date(payment_details.payment_
    date)"/>
                        </prompt>
                        <filled>
                              <if cond="confirm == 'yes'">
                                    <return namelist="payment_details"/>
                              <else/>
                                    <assign name="payment_details.payment_date"
    expr="undefined"/>
                                    <clear/>
                              </if>
                        </filled>
                        <catch event="help">
                              <prompt>
                                    <value
    expr="PromptHelp_Confirm_payment_details_payment_date()"/>
                              </prompt>
                        </catch>
                        <catch event="repeat">
                              <reprompt/>
                        </catch>

                  </field>

                  <catch event="help">
                        <prompt count="1" bargein="true" lq:text="">
                              <value
    expr="PromptHelp_Ask_payment_details_payment_date()"/>
                        </prompt>
                  </catch>

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

            </form>

            <form id="MENUITEM_main_quit" lq:element="MenuChoice"
    lq:name="quit">
                  <block lq:element="End">
                        <exit/>
                  </block>

            </form>

            <form id="BRANCH_checkPINvalidity_1" lq:element="Branch"
    lq:condition="otherwise">
                  <block name="link">
                        <goto next="#SPEAKER_failedPIN"/>
                  </block>

            </form>
```

```
        <form id="SPEAKER_failedPIN" lq:element="Speaker"
    lq:name="failedPIN">
                <block>
                      <prompt>
                            You have not been authenticated. Goodbye.
                      </prompt>
                </block>
                <block lq:element="End">
                      <exit/>
                </block>

        </form>

        <form id="LOOP_authenticate" lq:element="Loop"
    lq:name="authenticate" lq:testatstart="true">
                <var name="counter"/>
                <var name="get_PIN"/>
                <var name="validatePIN"/>
                <var name="call_get_balance"/>
                <var name="payment_details"/>
                <block>
                      <assign name="document.counter" expr="counter"/>
                      <assign name="document.get_PIN" expr="get_PIN"/>
                      <assign name="document.validatePIN"
    expr="validatePIN"/>
                      <assign name="document.call_get_balance"
    expr="call_get_balance"/>
                      <assign name="document.payment_details"
    expr="payment_details"/>
                </block>
                <block>
                      <goto next="#LOOPINIT_authenticate"/>
                </block>

        </form>

        <form id="LOOPINIT_authenticate" lq:references="authenticate">
                <block>
                      <assign name="counter" expr="0" lq:element="LoopInit"/>
                      <goto next="#LOOPTEST_authenticate"/>
                </block>
        </form>

        <form id="LOOPTEST_authenticate" lq:element="LoopTest"
    lq:test="counter&amp;lt;3" lq:references="authenticate">
                <block>
                      <if cond="counter&amp;lt;3">
                            <goto next="#LOOPBODY_authenticate"/>
                      <else/>
                            <goto next="#LOOPEND_authenticate"/>
                      </if>
                </block>

        </form>

        <form id="LOOPEND_authenticate" lq:element="LoopEnd"
    lq:references="authenticate">
```

```
                <block>
                        <prompt count="1" bargein="true" lq:text="">
                                <value expr="PromptLoopExit_authenticate()"/>
                        </prompt>
                        <return namelist="counter get_PIN validatePIN
    call_get_balance payment_details "/>
                </block>
        </form>

        <form id="LOOPSTEP_authenticate" lq:references="authenticate">
                <block>
                        <assign name="counter" expr="counter+1"
    lq:element="LoopStep"/>
                        <goto next="#LOOPTEST_authenticate"/>
                </block>
        </form>

        <form id="LOOPBODY_authenticate" lq:element="LoopBody"
    lq:references="authenticate">
                <block name="link">
                        <goto next="#FORM_get_PIN"/>
                </block>

        </form>

        <var name="get_PIN" expr="new Object()" lq:references="get_PIN"/>
        <form id="FORM_get_PIN" lq:element="Form" lq:name="get_PIN"
    lq:modal="true" lq:confirm="false" lq:documentGrammar="false">
                <field name="PIN" cond="false"/>
                <block>
                        <assign name="get_PIN.PIN" expr="PIN"/>
                </block>
                <subdialog name="get_PIN_PIN" src="#ASK_get_PIN_PIN">
                        <param name="get_PIN" expr="get_PIN"/>
                        <filled>
                                <assign name="get_PIN"
    expr="get_PIN_PIN.get_PIN"/>
                        </filled>
                </subdialog>

                <block name="link">
                        <goto next="#REMOTE_validatePIN"/>
                </block>

        </form>

        <form id="ASK_get_PIN_PIN" lq:element="FormField"
    lq:references="get_PIN" lq:name="PIN" lq:type="Digitstring" lq:default=""
    lq:confirm="false">
                <var name="get_PIN"/>

                <block>
                        <if cond="get_PIN.PIN != undefined">
                                <return namelist="get_PIN"/>
                        </if>
                </block>
```

```
<field name="PIN" modal="true">
      <link event="help">
            <grammar src="null#Help"/>
      </link>
      <link event="repeat">
            <grammar src="null#Repeat"/>
      </link>
      <grammar src="null#Ask_get_PIN_PIN"/>
      <prompt count="1" bargein="true" lq:text="please say
your four digit PIN number">
            <value expr="PromptAsk_get_PIN_PIN()"/>
      </prompt>
      <prompt count="2" bargein="true" lq:text="second
prompt. please say the pin. Example is one two three four.">
            <value expr="PromptAsk_get_PIN_PIN_2()"/>
      </prompt>
      <filled>
            <assign name="get_PIN.PIN"
expr="merge(get_PIN.PIN, PIN)"/>
      </filled>
</field>

<block>
      <return namelist="get_PIN"/>
</block>


<catch event="help">
      <prompt count="1" bargein="true" lq:text="">
            <value expr="PromptHelp_Ask_get_PIN_PIN()"/>
      </prompt>
</catch>

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

</form>

<var name="validatePIN" expr="new Object()"
lq:references="validatePIN"/>
<form id="REMOTE_validatePIN" lq:element="RemoteProcessing"
lq:name="validatePIN"
lq:src="http://videre.trl.telstra.com.au/~c797808/cgi-
bin/verify_pin.php">
      <var name="sending_pam" expr="get_PIN.PIN"
lq:element="Input"/>
      <subdialog name="remotecall"
src="http://videre.trl.telstra.com.au/~c797808/cgi-bin/verify_pin.php"
namelist="sending_pam ">
            <filled>
            <assign name="validatePIN.x" expr="remotecall.x"
lq:element="Output"/>
            </filled>
      </subdialog>
      <block name="link">
            <goto next="#DECISION_passOrFail"/>
```

```
                </block>

        </form>

        <form id="DECISION_passOrFail" lq:element="Decision"
lq:name="passOrFail">
                <block>
                        <if cond="validatePIN.x == &amp;apos;PASS&amp;apos;">
                                <goto next="#BRANCH_passOrFail_0"/>
                        <else/>
                                <goto next="#BRANCH_passOrFail_1"/>
                        </if>
                </block>
        </form>

        <form id="BRANCH_passOrFail_0" lq:element="Branch"
lq:condition="validatePIN.x == &amp;apos;PASS&amp;apos;">
                <block lq:element="Break">
                        <return namelist="counter get_PIN validatePIN
call_get_balance payment_details "/>
                </block>

        </form>

        <form id="BRANCH_passOrFail_1" lq:element="Branch"
lq:condition="otherwise">
                <block name="link">
                        <goto next="#SPEAKER_tryAgain"/>
                </block>

        </form>

        <form id="SPEAKER_tryAgain" lq:element="Speaker"
lq:name="tryAgain">
                <block>
                        <prompt>
                                Your PIN is not valid. Please try again.
                        </prompt>
                </block>
                <block lq:element="Next">
                        <goto next="#LOOPSTEP_authenticate"/>
                </block>

        </form>

</vxml>
```

FILE 2. SUBROUTINE1.VXML

```
<?xml version="1.0" encoding="utf-8"?>
<vxml version="1.0" xmlns:lq="http://www.telstra.com.au/LyreQuest">
      <link event="help">
            <grammar src="null#Help"/>
      </link>

      <link event="repeat">
            <grammar src="null#Repeat"/>
      </link>

      <link event="quit">
            <grammar src="null#Quit"/>
      </link>

      <script>
            function merge(obj1, obj2)
            {
                  if (obj1 == undefined)
                        return obj2;
                  else
                        return obj1;
            }
      </script>

      <var name="_dummy"/>

      <var name="customer_balance" lq:element="DocumentVariables"/>

      <var name="pin_number" lq:references="sub_present_balance"/>
      <var name="result_code" lq:references="sub_present_balance"/>
      <form id="SUBROUTINE_sub_present_balance" lq:element="Subroutine"
lq:name="sub_present_balance">
            <var name="pin_number" lq:element="Input"/>
            <var name="result_code" lq:element="Output"/>
            <block>
                  <assign name="document.pin_number" expr="pin_number"/>
                  <assign name="document.result_code"
expr="result_code"/>
            </block>
            <block name="link">
                  <goto next="#FORM_Form4"/>
            </block>

      </form>

      <var name="Form4" expr="new Object()" lq:references="Form4"/>
      <form id="FORM_Form4" lq:element="Form" lq:name="Form4"
lq:modal="false" lq:confirm="true" lq:documentGrammar="false">
            <field name="account_number" cond="false"/>
            <block>
                  <assign name="Form4.account_number"
expr="account_number"/>
```

```
                </block>
                <subdialog name="Form4_account_number"
    src="#ASK_Form4_account_number">
                      <param name="Form4" expr="Form4"/>
                      <filled>
                            <assign name="Form4"
    expr="Form4_account_number.Form4"/>
                      </filled>
                </subdialog>

                <field name="confirm">
                      <grammar src="null#Confirm"/>
                      <prompt>
                            <value
    expr="PromptConfirm_Form4(Form4.account_number)"/>
                      </prompt>
                      <filled>
                            <if cond="confirm != 'yes'">
                                  <assign name="Form4.account_number"
    expr="undefined"/>
                                  <clear/>
                            </if>
                      </filled>
                      <catch event="help">
                            <prompt>
                                  <value expr="PromptHelp_Confirm_Form4()"/>
                            </prompt>
                      </catch>
                      <catch event="repeat">
                            <reprompt/>
                      </catch>

                </field>
                <block name="link">
                      <goto next="#LOCAL_LocalProcessing5"/>
                </block>

          </form>

          <form id="ASK_Form4_account_number" lq:element="FormField"
    lq:references="Form4" lq:name="account_number" lq:type="Digitstring"
    lq:default="" lq:confirm="false">
                <grammar src="null#Ask_Form4_account_number"/>

    <var name="Form4"/>

    <block>
          <if cond="Form4.account_number != undefined">
                <return namelist="Form4"/>
          </if>
    </block>

    <field name="account_number" cond="false"/>
    <initial name="start">
          <prompt count="1" bargein="true" lq:text="">
                <value expr="PromptAsk_Form4_account_number()"/>
          </prompt>
```

```
                </initial>
                <filled mode="any">
                        <assign name="Form4.account_number"
    expr="merge(Form4.account_number, account_number)"/>
                        <if cond="Form4.account_number == undefined">
                                <clear/>
                        </if>
                </filled>

                <block>
                        <return namelist="Form4"/>
                </block>


                <catch event="help">
                        <prompt count="1" bargein="true" lq:text="">
                                <value
    expr="PromptHelp_Ask_Form4_account_number()"/>
                        </prompt>
                </catch>

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

        </form>

        <form id="LOCAL_LocalProcessing5" lq:element="LocalProcessing"
    lq:name="LocalProcessing5">
                <block>
                        <script>
                                <![CDATA[customer_balance = new Object();
    if (Form4.account_number ==  '12') {
        customer_balance.dollars = '1000';
        customer_balance.cents = '50';
    } else {
        customer_balance.dollars = '0';
        customer_balance.cents = '2';
    }
    result_code = pin_number + 1;]]>
                        </script>
                </block>

                <block name="link">
                        <goto next="#SPEAKER_Speaker6"/>
                </block>

        </form>

        <form id="SPEAKER_Speaker6" lq:element="Speaker"
    lq:name="Speaker6">
                <block>
                        <prompt>
                                your balance is
                                <value expr="PromptMoney(customer_balance)"/>
                        </prompt>
                </block>
```

```
                <block lq:element="Return">
                      <return namelist="result_code "/>
                </block>

        </form>

</vxml>
```

APPENDIX C: PORTION OF A SYSTEM GENERATED VoiceXML FILE

FILE 1. DEMO2_8MAY.VXML (INCOMPLETE, FOR EXAMPLE ONLY)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE vxml  PUBLIC "-//Nuance/DTD VoiceXML 1.2/EN"
"http://voicexml.nuance.com/dtd/nuancevoicexml-1-2.dtd">
<vxml version="1.0" application="demo2_8may.vxml">
      <link event="help">
            <grammar src="demo2_8may.gsl#Help"/>
      </link>

      <link event="repeat">
            <grammar src="demo2_8may.gsl#Repeat"/>
      </link>

      <link event="quit">
            <grammar src="demo2_8may.gsl#Quit"/>
      </link>

      <script>
            function merge(obj1, obj2)
            {
                  if (obj1 == undefined)
                        return obj2;
                  else
                        return obj1;
            }
      </script>

      <var name="_dummy"/>

      <var name="counter"/>

      <catch event="quit">
            <prompt count="1" bargein="true">
                  <value expr="PromptGoodbye()"/>
            </prompt>
            <exit/>
      </catch>

      <catch event="noinput">
            <prompt count="1" bargein="true">
                  <value expr="PromptNoinput()"/>
            </prompt>
            <reprompt/>
      </catch>

      <catch event="nomatch">
            <prompt count="1" bargein="true">
                  <value expr="PromptNomatch()"/>
            </prompt>
            <reprompt/>
```

```
</catch>

<form id="DIALOG_MainDialog">
      <block name="link">
            <goto next="#SPEAKER_sayWelcome"/>
      </block>

</form>

<form id="SPEAKER_sayWelcome">
      <block>
            <prompt>
                  Welcome to the automated X Y Z bank
            </prompt>
      </block>
      <block name="link">
            <goto next="#LOOPCALL_call_authenticate_loop"/>
      </block>

</form>

<form id="LOOPCALL_call_authenticate_loop">
      <subdialog name="loopcall" src="#LOOP_authenticate">
            <param name="counter" expr="document.counter"/>
            <param name="get_PIN" expr="document.get_PIN"/>
            <param name="validatePIN" expr="document.validatePIN"/>
            <param name="call_get_balance"
expr="document.call_get_balance"/>
            <param name="payment_details"
expr="document.payment_details"/>
            <filled>
                  <assign name="document.counter"
expr="loopcall.counter"/>
                  <assign name="document.get_PIN"
expr="loopcall.get_PIN"/>
                  <assign name="document.validatePIN"
expr="loopcall.validatePIN"/>
                  <assign name="document.call_get_balance"
expr="loopcall.call_get_balance"/>
                  <assign name="document.payment_details"
expr="loopcall.payment_details"/>
            </filled>
      </subdialog>
      <block name="link">
            <goto next="#DECISION_checkPINvalidity"/>
      </block>

</form>

<form id="DECISION_checkPINvalidity">
      <block>
            <if cond="validatePIN.x == &apos;PASS&apos;">
                  <goto next="#BRANCH_checkPINvalidity_0"/>
            <else/>
                  <goto next="#BRANCH_checkPINvalidity_1"/>
            </if>
      </block>
```

```
</form>

<form id="BRANCH_checkPINvalidity_0">
      <block name="link">
            <goto next="#MENU_main"/>
      </block>

</form>

<menu id="MENU_main">
      <prompt count="1" bargein="true">
            <value expr="PromptMenu_main()"/>
      </prompt>
      <choice next="#MENUITEM_main_balance">
            <grammar src="demo2_8may.gsl#Menu_main_balance"/>
      </choice>
      <choice next="#MENUITEM_main_enquiry">
            <grammar src="demo2_8may.gsl#Menu_main_enquiry"/>
      </choice>
      <choice next="#MENUITEM_main_leave_message">
            <grammar src="demo2_8may.gsl#Menu_main_leave_message"/>
      </choice>
      <choice next="#MENUITEM_main_pay_bills">
            <grammar src="demo2_8may.gsl#Menu_main_pay_bills"/>
      </choice>
      <choice next="#MENUITEM_main_quit">
            <grammar src="demo2_8may.gsl#Menu_main_quit"/>
      </choice>

      <catch event="help">
            <prompt count="1" bargein="true">
                  <value expr="PromptHelp_Menu_main()"/>
            </prompt>
      </catch>
      <catch event="repeat">
            <reprompt/>
      </catch>
</menu>

<form id="MENUITEM_main_balance">
      <block name="link">
            <goto next="#SUBROUTINECALL_call_get_balance"/>
      </block>

</form>

<var name="call_get_balance" expr="new Object()"/>
<form id="SUBROUTINECALL_call_get_balance">
      <subdialog name="subcall"
src="subroutine1.vxml#SUBROUTINE_sub_present_balance">
            <param name="pin_number" expr="get_PIN.PIN"/>
            <filled>
                  <assign name="call_get_balance.result_code"
expr="subcall.result_code"/>
            </filled>
      </subdialog>
      <block name="link">
```

```
                        <goto next="#SPEAKER_Speaker24"/>
                  </block>

            </form>

            <form id="SPEAKER_Speaker24">
                  <block>
                        <prompt>
                              the result of your balance enquiry is
                              <value expr="call_get_balance.result_code"/>
                        </prompt>
                  </block>
                  <block>
                        <goto next="#MENU_main"/>
                  </block>

            </form>

            <form id="MENUITEM_main_enquiry">
                  <block name="link">
                        <goto next="#SPEAKER_Speaker21"/>
                  </block>

            </form>

            <form id="SPEAKER_Speaker21">
                  <block>
                        <prompt>
                              you are being transfered to an operator
                        </prompt>
                  </block>
                  <block name="link">
                        <goto next="#TRANSFER_Transfer22"/>
                  </block>

            </form>

            <form id="TRANSFER_Transfer22">
                  <transfer name="Transfer22" bridge="false"
connecttimeout="30" destexpr="'tel:0392536742'">
                  </transfer>
            </form>

            <form id="MENUITEM_main_leave_message">
                  <block name="link">
                        <goto next="#RECORD_Record20"/>
                  </block>

            </form>

            <var name="Record20"/>
            <form id="RECORD_Record20">
                  <record name="record_Record20" beep="true" maxtime="30s"
dtmfterm="true">
                        <prompt count="1" bargein="true">
                              <value expr="PromptRecord_Record20()"/>
                        </prompt>
```

```
                </record>

                <field name="confirm" modal="true">
                      <grammar src="demo2_8may.gsl#Confirm"/>
                      <prompt>
                            Your recording is <value expr="record_Record20"/>
    To keep it, say yes. To discard it, say no.
                      </prompt>
                      <filled>
                            <if cond="confirm != 'yes'">
                                  <clear/>
                            </if>
                      </filled>
                      <catch event="help">
                            <prompt>
                                  Please say yes to keep your recording, or
    say no to record another message.
                            </prompt>
                      </catch>
                      <catch event="repeat">
                            <prompt>
                                  Your recording is <value
    expr="record_Record20"/> To keep it, say yes. To discard it, say no.
                            </prompt>
                      </catch>
                </field>

                <block>
                      <assign name="Record20" expr="record_Record20"/>
                </block>

                <block>
                      <goto next="#MENU_main"/>
                </block>

          </form>

          <form id="MENUITEM_main_pay_bills">
                <block name="link">
                      <goto next="#FORM_payment_details"/>
                </block>

          </form>

          <var name="payment_details" expr="new Object()"/>
          <form id="FORM_payment_details">
                <grammar scope="document"
    src="demo2_8may.gsl#Form_payment_details"/>

                <field name="company" cond="false"/>
                <field name="amount" cond="false"/>
                <field name="payment_date" cond="false"/>
                <block>
                      <assign name="payment_details.company" expr="company"/>
                      <assign name="payment_details.amount" expr="amount"/>
                      <assign name="payment_details.payment_date"
    expr="payment_date"/>
```

```
                </block>
                <subdialog name="payment_details_company"
    src="#ASK_payment_details_company">
                      <param name="payment_details" expr="payment_details"/>
                      <filled>
                            <assign name="payment_details"
    expr="payment_details_company.payment_details"/>
                      </filled>
                </subdialog>

                <subdialog name="payment_details_amount"
    src="#ASK_payment_details_amount">
                      <param name="payment_details" expr="payment_details"/>
                      <filled>
                            <assign name="payment_details"
    expr="payment_details_amount.payment_details"/>
                      </filled>
                </subdialog>

                <block>
                      <if cond="payment_details.payment_date == undefined">
                            <script>
                                  <![CDATA[
                                        payment_details.payment_date = new
    Object();
                                        payment_details.payment_date.modifier
    = 'today';
                                  ]]>
                            </script>
                      </if>
                </block>

                <subdialog name="payment_details_payment_date"
    src="#ASK_payment_details_payment_date">
                      <param name="payment_details" expr="payment_details"/>
                      <filled>
                            <assign name="payment_details"
    expr="payment_details_payment_date.payment_details"/>
                      </filled>
                </subdialog>

                <field name="confirm">
                      <grammar src="demo2_8may.gsl#Confirm"/>
                      <prompt>
                            <value
    expr="PromptConfirm_payment_details(payment_details.company,
    payment_details.amount, payment_details.payment_date)"/>
                      </prompt>
                      <filled>
                            <if cond="confirm != 'yes'">
                                  <assign name="payment_details.company"
    expr="undefined"/>
                                  <assign name="payment_details.amount"
    expr="undefined"/>
                                  <assign name="payment_details.payment_date"
    expr="undefined"/>
                                  <clear/>
```

```
                        </if>
                    </filled>
                    <catch event="help">
                        <prompt>
                            <value
expr="PromptHelp_Confirm_payment_details()"/>
                        </prompt>
                    </catch>
                    <catch event="repeat">
                        <reprompt/>
                    </catch>

                </field>
                <block>
                    <goto next="#MENU_main"/>
                </block>

            </form>

            <form id="ASK_payment_details_company">
                <grammar src="demo2_8may.gsl#Ask_payment_details_company"/>

                <var name="payment_details"/>

                <block>
                    <if cond="payment_details.company != undefined">
                        <return namelist="payment_details"/>
                    </if>
                </block>

                <field name="company" cond="false"/>
                <field name="amount" cond="false"/>
                <field name="payment_date" cond="false"/>
                <initial name="start">
                    <prompt count="1" bargein="true">
                        <value
expr="PromptAsk_payment_details_company()"/>
                    </prompt>
                </initial>
                <filled mode="any">
                    <assign name="payment_details.company"
expr="merge(payment_details.company, company)"/>
                    <assign name="payment_details.amount"
expr="merge(payment_details.amount, amount)"/>
                    <assign name="payment_details.payment_date"
expr="merge(payment_details.payment_date, payment_date)"/>
                    <if cond="payment_details.company == undefined">
                        <clear/>
                    </if>
                </filled>

                <block>
                    <return namelist="payment_details"/>
                </block>

                <catch event="help">
```

```
                        <prompt count="1" bargein="true">
                              <value
    expr="PromptHelp_Ask_payment_details_company()"/>
                        </prompt>
                  </catch>

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

            </form>

            <form id="ASK_payment_details_amount">
                  <grammar src="demo2_8may.gsl#Ask_payment_details_amount"/>

                  <var name="payment_details"/>

                  <block>
                        <if cond="payment_details.amount != undefined">
                              <return namelist="payment_details"/>
                        </if>
                  </block>

                  <field name="amount" cond="false"/>
                  <field name="payment_date" cond="false"/>
                  <initial name="start">
                        <prompt count="1" bargein="true">
                              <value
    expr="PromptAsk_payment_details_amount()"/>
                        </prompt>
                  </initial>
                  <filled mode="any">
                        <assign name="payment_details.amount"
    expr="merge(payment_details.amount, amount)"/>
                        <assign name="payment_details.payment_date"
    expr="merge(payment_details.payment_date, payment_date)"/>
                        <if cond="payment_details.amount == undefined">
                              <clear/>
                        </if>
                  </filled>

                  <block>
                        <return namelist="payment_details"/>
                  </block>

                  <catch event="help">
                        <prompt count="1" bargein="true">
                              <value
    expr="PromptHelp_Ask_payment_details_amount()"/>
                        </prompt>
                  </catch>

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

```
        </form>

        <form id="ASK_payment_details_payment_date">
              <grammar
src="demo2_8may.gsl#Ask_payment_details_payment_date"/>

              <var name="payment_details"/>

              <block>
                    <if cond="payment_details.payment_date != undefined">
                          <assign name="start" expr="true"/>
                          <goto nextitem="confirm"/>
                    </if>
              </block>

              <field name="payment_date" cond="false"/>
              <initial name="start">
                    <prompt count="1" bargein="true">
                          <value
expr="PromptAsk_payment_details_payment_date()"/>
                    </prompt>
              </initial>
              <filled mode="any">
                    <assign name="payment_details.payment_date"
expr="merge(payment_details.payment_date, payment_date)"/>
                    <if cond="payment_details.payment_date == undefined">
                          <clear/>
                    </if>
              </filled>

              <field name="confirm">
                    <grammar src="demo2_8may.gsl#Confirm"/>
                    <prompt>
                          <value
expr="PromptConfirm_payment_details_payment_date(payment_details.payment_
date)"/>
                    </prompt>
                    <filled>
                          <if cond="confirm == 'yes'">
                                <return namelist="payment_details"/>
                          <else/>
                                <assign name="payment_details.payment_date"
expr="undefined"/>
                                <clear/>
                          </if>
                    </filled>
                    <catch event="help">
                          <prompt>
                                <value
expr="PromptHelp_Confirm_payment_details_payment_date()"/>
                          </prompt>
                    </catch>
                    <catch event="repeat">
                          <reprompt/>
                    </catch>

              </field>
```

```
                <catch event="help">
                        <prompt count="1" bargein="true">
                                <value
expr="PromptHelp_Ask_payment_details_payment_date()"/>
                        </prompt>
                </catch>

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

        </form>

        <form id="MENUITEM_main_quit">
                <block>
                        <exit/>
                </block>

        </form>

        <form id="BRANCH_checkPINvalidity_1">
                <block name="link">
                        <goto next="#SPEAKER_failedPIN"/>
                </block>

        </form>

        <form id="SPEAKER_failedPIN">
                <block>
                        <prompt>
                                You have not been authenticated. Goodbye.
                        </prompt>
                </block>
                <block>
                        <exit/>
                </block>

        </form>

        <form id="LOOP_authenticate">
                <var name="counter"/>
                <var name="get_PIN"/>
                <var name="validatePIN"/>
                <var name="call_get_balance"/>
                <var name="payment_details"/>
                <block>
                        <assign name="document.counter" expr="counter"/>
                        <assign name="document.get_PIN" expr="get_PIN"/>
                        <assign name="document.validatePIN"
expr="validatePIN"/>
                        <assign name="document.call_get_balance"
expr="call_get_balance"/>
                        <assign name="document.payment_details"
expr="payment_details"/>
                </block>
                <block>
```

```
                        <goto next="#LOOPINIT_authenticate"/>
                  </block>

            </form>

            <form id="LOOPINIT_authenticate">
                  <block>
                        <assign name="counter" expr="0"/>
                        <goto next="#LOOPTEST_authenticate"/>
                  </block>
            </form>

            <form id="LOOPTEST_authenticate">
                  <block>
                        <if cond="counter&lt;3">
                              <goto next="#LOOPBODY_authenticate"/>
                        <else/>
                              <goto next="#LOOPEND_authenticate"/>
                        </if>
                  </block>

            </form>

            <form id="LOOPEND_authenticate">
                  <block>
                        <prompt count="1" bargein="true">
                              <value expr="PromptLoopExit_authenticate()"/>
                        </prompt>
                        <return namelist="counter get_PIN validatePIN
call_get_balance payment_details "/>
                  </block>
            </form>

            <form id="LOOPSTEP_authenticate">
                  <block>
                        <assign name="counter" expr="counter+1"/>
                        <goto next="#LOOPTEST_authenticate"/>
                  </block>
            </form>

            <form id="LOOPBODY_authenticate">
                  <block name="link">
                        <goto next="#FORM_get_PIN"/>
                  </block>

            </form>

            <var name="get_PIN" expr="new Object()"/>
            <form id="FORM_get_PIN">
                  <field name="PIN" cond="false"/>
                  <block>
                        <assign name="get_PIN.PIN" expr="PIN"/>
                  </block>
                  <subdialog name="get_PIN_PIN" src="#ASK_get_PIN_PIN">
                        <param name="get_PIN" expr="get_PIN"/>
                        <filled>
```

```
                        <assign name="get_PIN"
    expr="get_PIN_PIN.get_PIN"/>
                    </filled>
                </subdialog>

                <block name="link">
                    <goto next="#REMOTE_validatePIN"/>
                </block>

        </form>

        <form id="ASK_get_PIN_PIN">
            <var name="get_PIN"/>

            <block>
                <if cond="get_PIN.PIN != undefined">
                    <return namelist="get_PIN"/>
                </if>
            </block>

            <field name="PIN" modal="true">
                <link event="help">
                    <grammar src="demo2_8may.gsl#Help"/>
                </link>
                <link event="repeat">
                    <grammar src="demo2_8may.gsl#Repeat"/>
                </link>
                <grammar src="demo2_8may.gsl#Ask_get_PIN_PIN"/>
                <prompt count="1" bargein="true">
                    <value expr="PromptAsk_get_PIN_PIN()"/>
                </prompt>
                <prompt count="2" bargein="true">
                    <value expr="PromptAsk_get_PIN_PIN_2()"/>
                </prompt>
                <filled>
                    <assign name="get_PIN.PIN"
    expr="merge(get_PIN.PIN, PIN)"/>
                </filled>
            </field>

            <block>
                <return namelist="get_PIN"/>
            </block>

            <catch event="help">
                <prompt count="1" bargein="true">
                    <value expr="PromptHelp_Ask_get_PIN_PIN()"/>
                </prompt>
            </catch>

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

        </form>
```

```
        <var name="validatePIN" expr="new Object()"/>
        <form id="REMOTE_validatePIN">
              <var name="sending_pam" expr="get_PIN.PIN"/>
              <subdialog name="remotecall"
src="http://videre.trl.telstra.com.au/~c797808/cgi-bin/verify_pin.php"
namelist="sending_pam ">
                    <filled>
                    <assign name="validatePIN.x" expr="remotecall.x"/>
                    </filled>
              </subdialog>
              <block name="link">
                    <goto next="#DECISION_passOrFail"/>
              </block>

        </form>

        <form id="DECISION_passOrFail">
              <block>
                    <if cond="validatePIN.x == &apos;PASS&apos;">
                          <goto next="#BRANCH_passOrFail_0"/>
                    <else/>
                          <goto next="#BRANCH_passOrFail_1"/>
                    </if>
              </block>
        </form>

        <form id="BRANCH_passOrFail_0">
              <block>
                    <return namelist="counter get_PIN validatePIN
call_get_balance payment_details "/>
              </block>

        </form>

        <form id="BRANCH_passOrFail_1">
              <block name="link">
                    <goto next="#SPEAKER_tryAgain"/>
              </block>

        </form>

        <form id="SPEAKER_tryAgain">
              <block>
                    <prompt>
                          Your PIN is not valid. Please try again.
                    </prompt>
              </block>
              <block>
                    <goto next="#LOOPSTEP_authenticate"/>
              </block>

        </form>

    <script><![CDATA[

    ...
```

```
    function PromptHelp_Ask_get_PIN_PIN() {
        var result;
        result = "please say the pin. for example, ten eighty six ";
           if( valid_string(result)){
                return result;
           }
     ;
        return result;
    }

    function PromptHelp_Confirm_payment_details_payment_date() {
        var result;
        result = "please say yes or no ";
           if( valid_string(result)){
                return result;
           }
     ;
        return result;
    }

    function Confirmation(confirmation) {
        var result;
        if(
          (confirmation == "yes" )
        ) {
           result = "true ";
           if( valid_string(result)){
                return result;
           }
        };

        if(
          (confirmation == "no" )
        ) {
           result = "false ";
           if( valid_string(result)){
                return result;
           }
        };

        result = confirmation + " ";
        return result;
    }

    function PromptConfirm_payment_details(company,amount,payment_date) {
        var result;
        result = "the company is " + Bpay_names(company)  + "the amount is "
    + Money(amount)  + "the payment date is " + Date(payment_date)  + "is
    this correct? ";
           if( valid_string(result)){
                return result;
           }
     ;
        return result;
    }

    …
```

```
function PromptConfirm_payment_details_payment_date(payment_date) {
    var result;
    result = "the payment date is " + Date(payment_date)  + "is this
correct? ";
       if( valid_string(result)){
            return result;
       }
 ;
    return result;
}

function PromptMenu_main_leave_message() {
    var result;
    result = "please say one of balance, enquiry, leave message, pay
bills or quit (note: please respond with another phrase that means 'leave
message') ";
       if( valid_string(result)){
            return result;
       }
 ;
    return result;
}


function Integernum(integer) {
function PromptAsk_payment_details_amount() {
    var result;
    result = "please say the amount ";
       if( valid_string(result)){
            return result;
       }
 ;
    return result;
}

function Bpay_names(bpay_names) {
    var result;
    if(
      (bpay_names == "telstra" )
    ) {
       result = "telstra ";
       if( valid_string(result)){
            return result;
       }
    };

    if(
      (bpay_names == "origin energy" )
    ) {
       result = "origin energy ";
       if( valid_string(result)){
            return result;
       }
    };

    if(
```

```
        (bpay_names == "citipower" )
    ) {
        result = "citipower ";
        if( valid_string(result)){
            return result;
        }
    };

    result = bpay_names + " ";
    return result;
}

function PromptHelp_Ask_payment_details_amount() {
    var result;
    result = "please say the amount. for example, three dollars and fifty
cents ";
        if( valid_string(result)){
            return result;
        }
 ;
    return result;
}

function PromptMenu_main() {
    var result;
    result = "please say one of balance, enquiry, leave message, pay
bills or quit ";
        if( valid_string(result)){
            return result;
        }
 ;
    return result;
}

function PromptConfirmation(confirmation) {
    var result;
    result = Confirmation(confirmation) ;
        if( valid_string(result)){
            return result;
        }
 ;
    return result;
}

function PromptHelp_Ask_payment_details_payment_date() {
    var result;
    result = "please say the payment date. for example, january the fifth
nineteen ninety nine ";
        if( valid_string(result)){
            return result;
        }
 ;
    return result;
}

function AM() {
    var result;
```

```
        result = "in the morning ";
           if( valid_string(result)){
                return result;
           }
;
        return result;
    }
    function PM() {
        var result;
        result = "in the " + EVENING() ;
           if( valid_string(result)){
                return result;
           }
 ;
        return result;
    }

    function PromptInteger(integer) {
        var result;
        result = Integer(integer) ;
           if( valid_string(result)){
                return result;
           }
 ;
        return result;
    }

    …

]]></script>
</vxml>
```

CLAIMS:

1. A process for developing a voice application, including:

generating graphical user interface components for selecting and interconnecting graphical icons representing dialog elements for voice applications, said dialog elements representing functional components of voice applications, having user configurable properties, and corresponding to respective predetermined sequences of VoiceXML elements;

receiving user input generated by user interaction with said graphical user interface components, said user input representing selection, arrangement, and interconnections of ones of said graphical icons, and configuration of dialog elements corresponding to ones of the selected graphical icons;

processing said user input to define a voice application by selecting dialog elements corresponding to the selected ones of said icons, configuring properties of ones of the selected dialog elements, and defining execution flows of said voice application from the interconnections of the selected icons; and

generating application code to provide said voice application;

wherein

the selected icons are interconnected to define a tree structure of execution flows for the voice application, each path through said tree structure representing a corresponding one of said execution flows; and

said application code represents each dialog element of said application as a linked sequence of VoiceXML elements including extended attributes that allows the tree structure of execution flows to be determined from said application code.

2. A process as claimed in claim 1, wherein said extended attributes allow the selected dialog elements and configured properties to be determined from said application code.

3. A process as claimed in claim 1 or 2, wherein said extended attributes are qualified names of a qualified XML namespace.

4. A process as claimed in any one of claims 1 to 3, wherein each dialog element of said application code includes a reference to the next of said dialog elements in an execution flow of said application to provide the linked sequence.

5. A process as claimed in any one of claims 1 to 4, including processing said application code to generate a visual representation of said dialog elements and said execution flows.

6. A process as claimed in any one of claims 1 to 5, wherein said step of generating application code includes generating extended VoiceXML code, prompt data, and grammar data for said application.

7. A process as claimed in claim 6, wherein said prompt data is represented as a grammar, and said process includes improving said grammar.

8. A process as claimed in claim 6 or 7, including generating at least one script for generating a prompt for said application on the basis of one or more parameters supplied to said script.

9. A process as claimed in claim 8, wherein said at least one script is generated on the basis of at least one script template and prompt data defined for said prompt by a user.

10. A process as claimed in claim 8 or 9, wherein said at least one script includes ECMAscript.

11. A process as claimed in any one of claims 6 to 10, including generating VoiceXML code and IVR grammar data for execution of said application on an IVR system on the basis of said extended VoiceXML code, prompt data, and grammar data.

12. A system having components for executing a process as claimed in any one of claims 1 to 11.

13. A computer readable storage medium having stored thereon a voice application generated by executing a process as claimed in any one of claims 1 to 11.

14. A computer readable storage medium having stored thereon executable programming instructions that, when executed by at least one computer processor, cause the processor to execute a process as claimed in any of claims 1 to 11.

15. A system for use in developing a voice application, including:

a dialog element selector for defining execution flows of a voice application by selecting and interconnecting graphical icons representing dialog elements for said voice application, said dialog elements representing functional components of said voice application, having user configurable properties, and corresponding to respective predetermined sequences of VoiceXML elements;

means for receiving user input generated by user interaction with said dialog element selector, said user input representing selection, arrangement, and interconnections of ones of said graphical icons, and configuration of dialog elements corresponding to ones of the selected graphical icons;

means for processing said user input to define a voice application by selecting dialog elements corresponding to the selected ones of said icons, configuring properties of ones of the selected dialog elements, and defining execution flows of said voice application from the interconnections of said icons; and

a code generator for generating application code to provide said voice application;

wherein

the interconnections of the selected icons define a tree structure of execution flows for the voice application, each path through said tree structure representing a corresponding one of said execution flows; and

said application code represents each dialog element of said application as a linked sequence of VoiceXML elements including one or more extended attributes that allow the tree structure of execution flows to be determined from said application code.

16. A system as claimed in claim 15, wherein said extended attributes allow the selected dialog elements and configured properties to be determined from said application code.

17. A system as claimed in claim 15 or 16, wherein said selector processes said application code to generate a graphical representation of said dialog elements and said execution flows of said voice application.

18. A system as claimed in any one of claims 15 to 17, wherein said code generator generates extended VoiceXML code, prompt data, and grammar data for said application.

19. A system as claimed in claim 18, wherein said prompt data is represented as a grammar, and the system includes one or more modules for improving said grammar.

20. A system as claimed in claim 18 or 19, including a script generator for generating at least one script for generating a prompt for said application on the basis of one or more parameters supplied to said script.

21. A system as claimed in claim 20, wherein said script generator generates said at least one script on the basis of at least one script template and prompt data defined for said prompt by a user.

22. A system as claimed in any one of claims 18 to 21, wherein said code generator generates VoiceXML code and IVR grammar data for execution of said application on an IVR system on the basis of said extended VoiceXML code, prompt data, and grammar data.

23. A computer readable storage medium having stored thereon an extended VoiceXML file generated by a system as claimed in any one of claims 15 to 22.

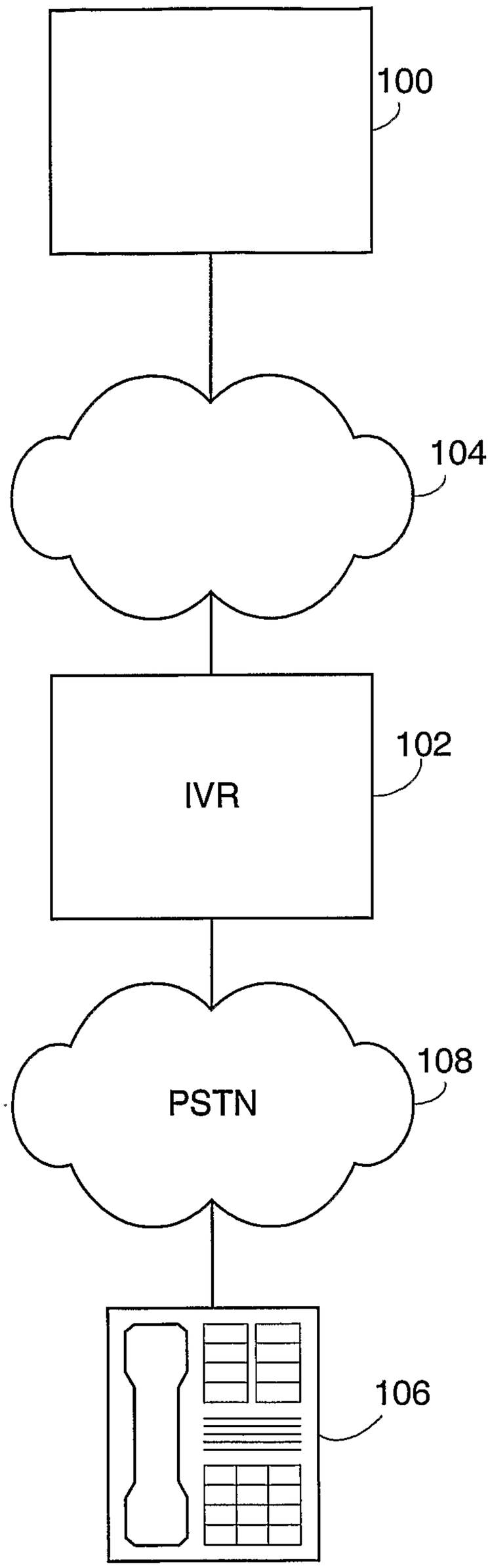


Figure 1

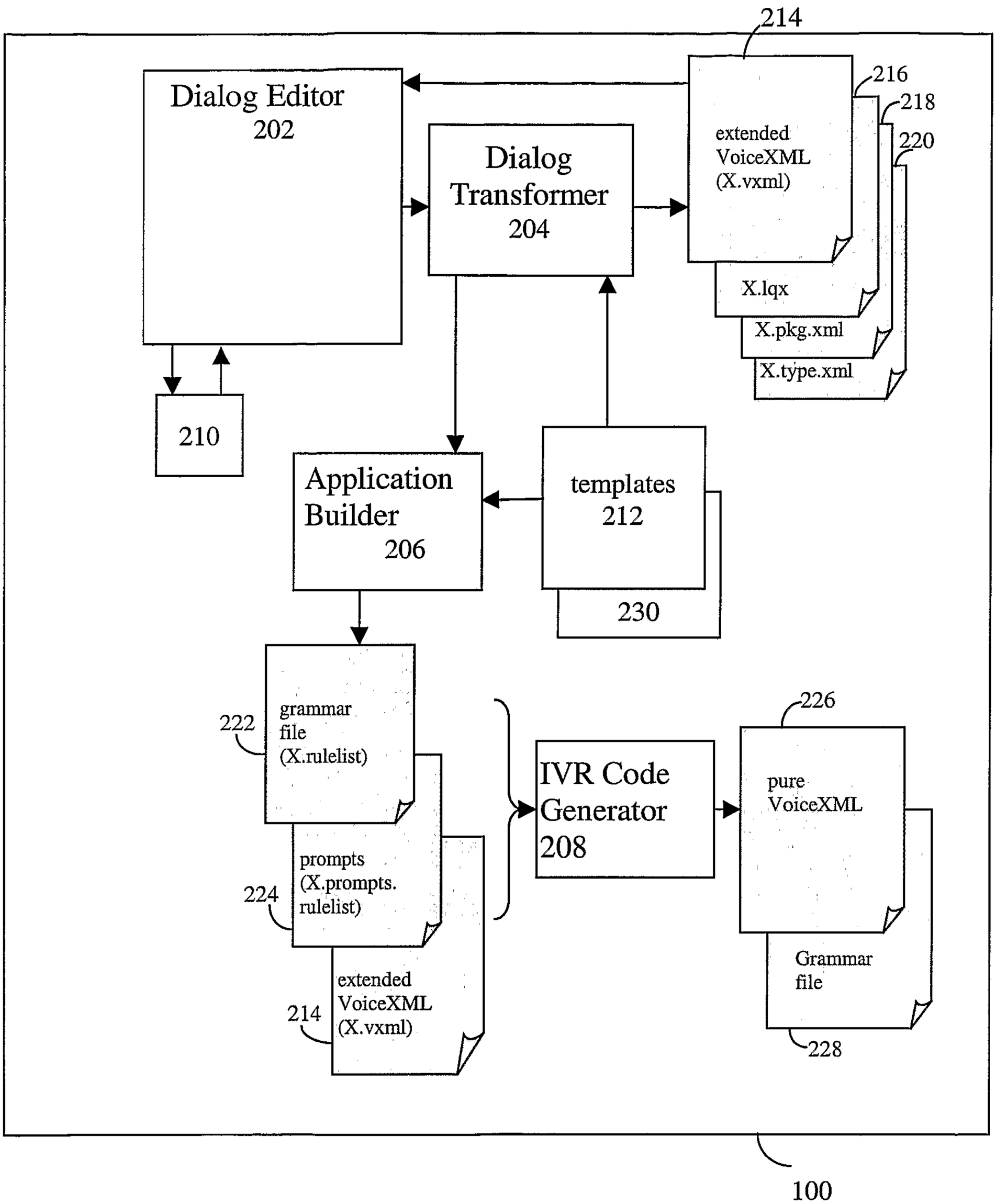

214
Dialog Editor
202
Dialog
Transformer
204
extended
VoiceXML
(X.vxml)
216
218
220
X.lqx
X.pkg.xml
X.type.xml
210
Application
Builder
206
templates
212
230
grammar
file
(X.rulelist)
222
prompts
(X.prompts.
rulelist)
224
extended
VoiceXML
(X.vxml)
214
IVR Code
Generator
208
pure
VoiceXML
226
Grammar
file
228
100

Figure 2

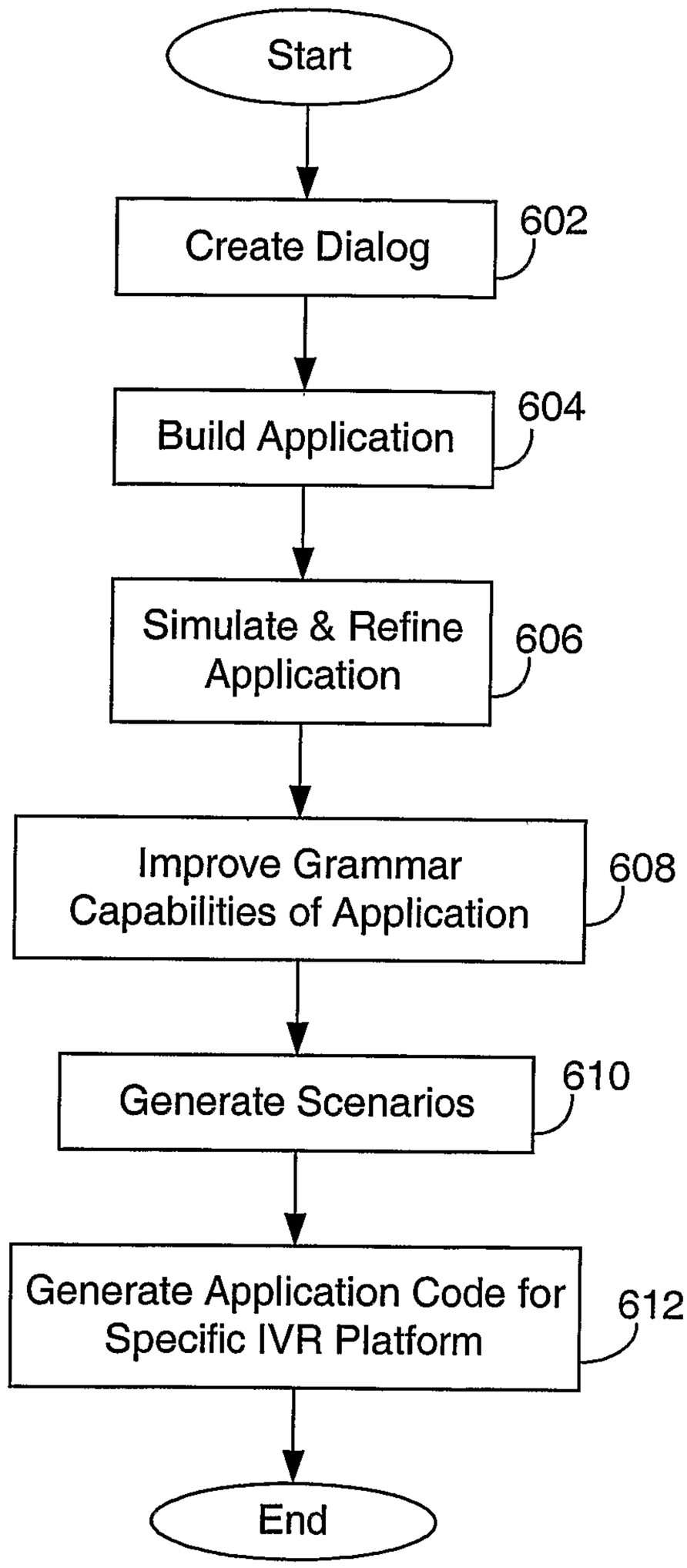

Start
Create Dialog
602
Build Application
604
Simulate & Refine Application
606
Improve Grammar Capabilities of Application
608
Generate Scenarios
610
Generate Application Code for Specific IVR Platform
612
End

Figure 3

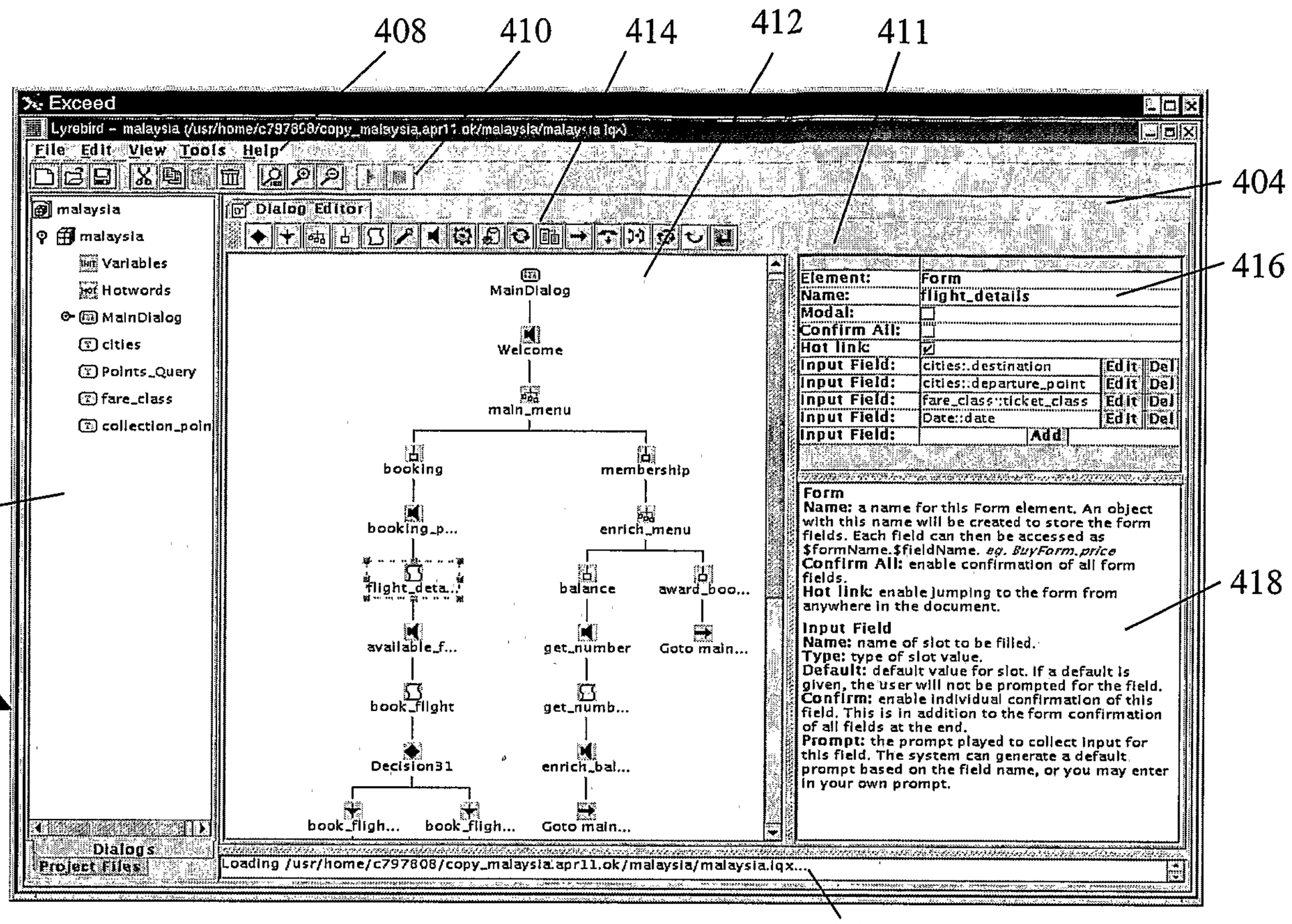


Figure 4

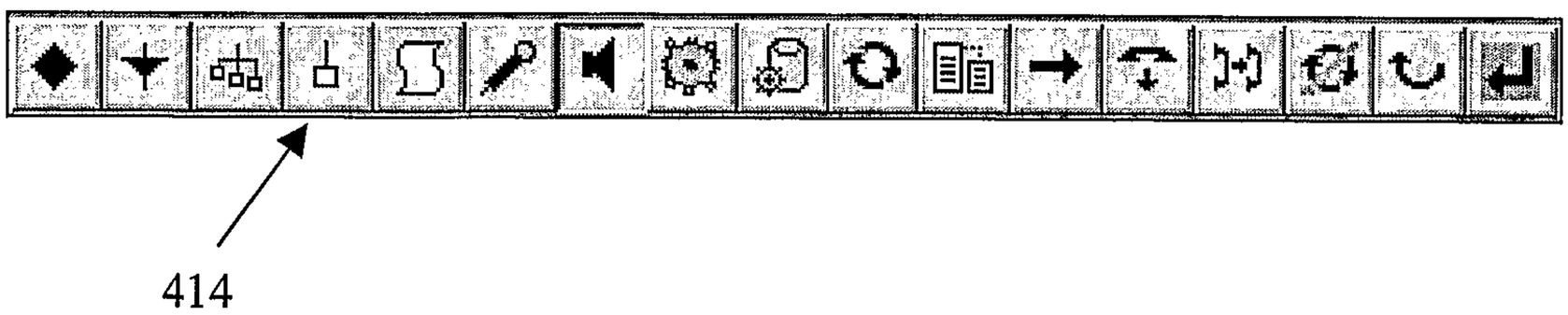


Figure 5

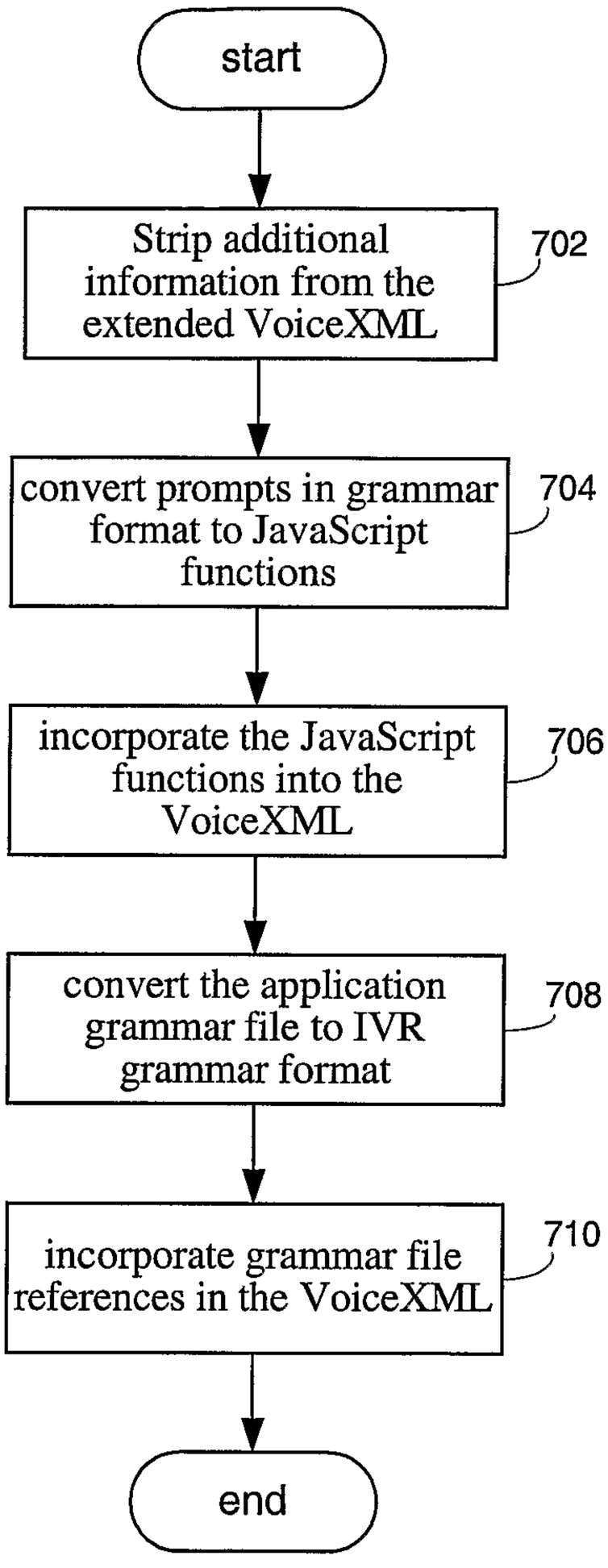
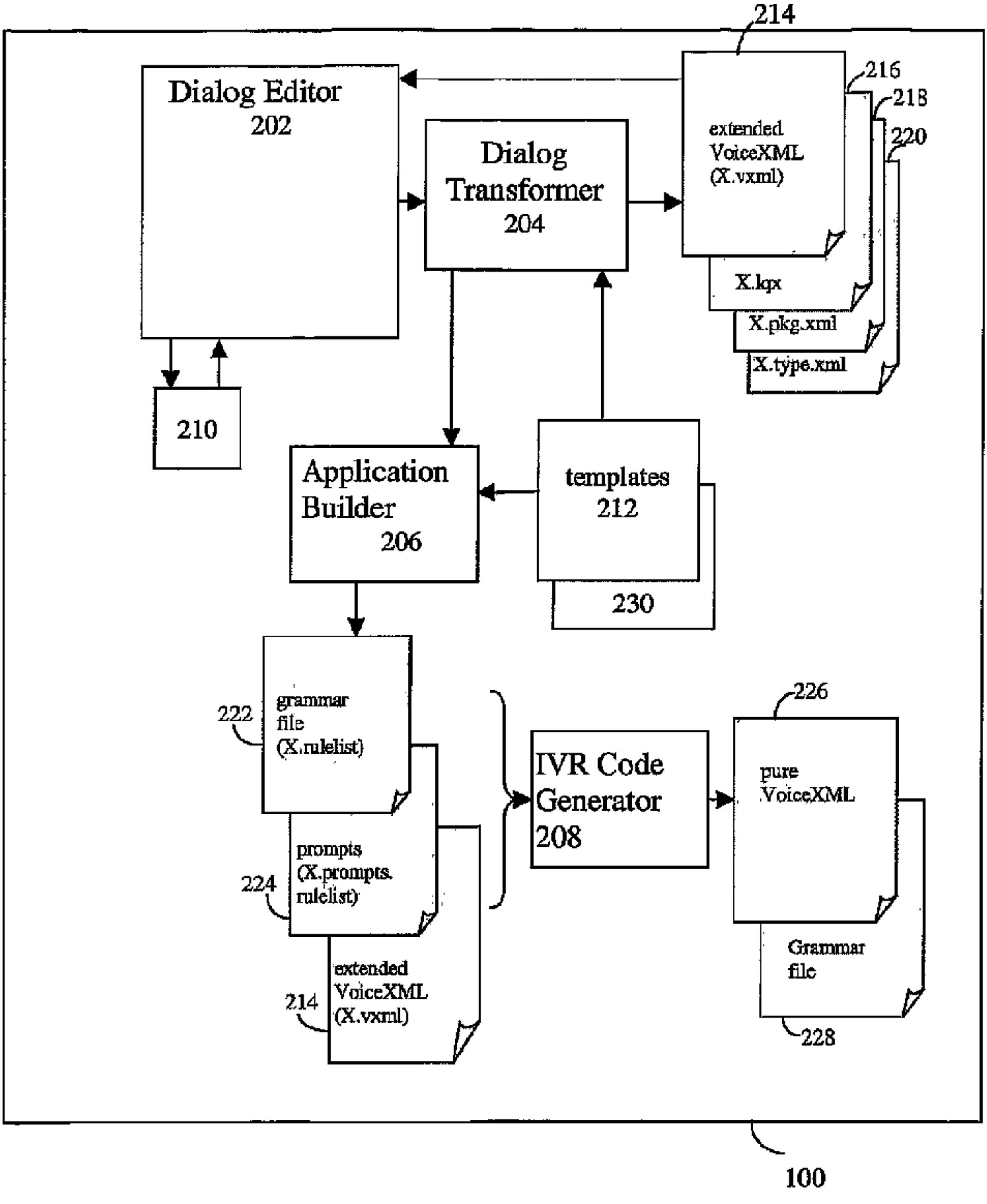

start
Strip additional information from the extended VoiceXML
702
convert prompts in grammar format to JavaScript functions
704
incorporate the JavaScript functions into the VoiceXML
706
convert the application grammar file to IVR grammar format
708
incorporate grammar file references in the VoiceXML
710
end

Figure 6


Dialog Editor
202
Dialog
Transformer
204
214
extended
VoiceXML
(X.vxml)
216
218
220
X.lqx
X.pkg.xml
X.type.xml
210
Application
Builder
206
templates
212
230
222
grammar
file
(X.rulelist)
IVR Code
Generator
208
226
pure
VoiceXML
224
prompts
(X.prompts.
rulelist)
214
extended
VoiceXML
(X.vxml)
Grammar
file
228
100