Mobile Visual Access to Legacy Voice-based Applications

Marco Avvenuti
Dip. di Ingegneria dell’Informazione
Università di Pisa
via Diotisalvi 2
56122 Pisa, Italy
m.avvenuti@iet.unipi.it

Alessio Vecchio
Dip. di Ingegneria dell’Informazione
Università di Pisa
via Diotisalvi 2
56122 Pisa, Italy
a.vecchio@ing.unipi.it

ABSTRACT
Vocal systems are a well-established reality. Examples of popular voice-based applications include helpdesk services, reservation systems, and vocal mailboxes. These services, that are accessible by mobile or fixed phones, suffer from some drawbacks: they cannot be used by people with hearing impairments, and they are not user-friendly for recurring customers who are forced into a bothersome interaction model. Thanks to the current availability of mobile devices with high processing capability and large memory, it is now possible to augment vocal systems with a graphical presentation. We designed an interpreter that converts the interaction between the user and the applications from vocal to visual, improving the usability of services as well as their accessibility. This has been achieved without requiring any substantial modification to the original voice-based systems. In this way, the service providers are not forced to re-write the applications or even abandon the technology in which they had invested.

Categories and Subject Descriptors
D.2.11 [Software]: Software Architectures; H.5.2 [Information Interfaces and Presentation]: User Interfaces—Graphical user interfaces (GUI); H.1.2 [Models and Principles]: User/Machine Systems—Human factors

General Terms
Design, Human Factors

Keywords
VoiceXML, Mobile devices, Android, Java Micro Edition

1. INTRODUCTION
In the last two decades we assisted to an always growing synergy between telecommunications and information technology. In particular, the computer telephony integration allows to embed data processing and intelligence within a telephonic system. Common application of computer telephony (CT) includes customer assistance, automated reservation systems, and vocal messaging. The CT market is extremely relevant in terms of both available applications and installed hardware devices (such as interactive voice response systems, automatic call distributors, and private branch exchanges).

An interactive voice response (IVR) system automates the management of phone calls by interacting with one or more databases. These systems are composed by a subsystem that guides the user within a set of vocal menus, generated through text-to-speech technology (TTS), and a subsystem that receives and executes the choices made by the user. Such choices can be expressed through the DTMF keypad or by means of vocal commands. In the last case, an automatic speech recognition (ASR) system is also needed. The use of IVR systems grew exponentially in the last few years, because companies trust it as a way to support costumers with reduced costs (they are obviously less expensive, more scalable and available with respect to similar human-based services).

Despite their numerous advantages, vocal systems suffer from few problems that limit their usage. In fact, sometimes, users feel that such applications are scarcely user-friendly, mostly because of long periods where they are forced to listen to information that is already known or not relevant for them. In other words, the lack of control tools in the hand of users makes them unable to decide how the execution of the vocal application should be articulated. For instance, let us suppose that a user is listening to a vocal menu and that he wants to select the tenth item. The user is forced to listen all the preceding items even if he is not interested. Another example is represented by a reservation system that, before starting the real reservation phase, includes a long list of requirements and/or constraints: if such service is accessed more than once, the user is forced to waste his time for receiving information that he already knows. A second, but maybe even more important, issue of IVR-based systems is that they cannot be accessed by hearing-impaired people. For these people, the company that supplies the service should provide a replicated version of the application through a different channel, such as the Web. This obviously increases the costs of the application development.

As mentioned, IVR-based technologies and systems are strongly consolidated and there is an extremely large number of vocal applications already in use. It is therefore unpractical, or economically inconvenient, to modify existing systems in order to cope with the previously mentioned problems. We conceived and designed an extension of IVR-based architectures that enhances such services and extends the user base. Our extension, that requires limited changes to existing systems and applications, is based on the idea of enabling access also through mobile devices such as smart-
phones and PDAs. Interaction between the user and the system is no longer restricted to vocal dialogs, but it can also take place by means of a graphical user interface displayed onto the user’s device. In particular, we designed and implemented VOXI, a VoiceXML interpreter that is executed on the user’s device: vocal menus are transformed into a list of graphical elements where the user can directly access the item of interest; repetitive information, that is now visualized as text, can be skipped; every function can be accessed by hearing-impaired.

2. VOICE-BASED SYSTEMS AND APPLICATIONS

In this section we briefly describe the architecture of traditional voice-based systems and the structure of applications. We also introduce the basic concepts of some related technologies.

When developing a vocal application, the programmer designs the call flow by wiring a number of basic elements provided by a library. Once the application has been created, it is transferred and installed into the server (Figure 1). In this context, the de-facto standard language used for development is VoiceXML, and applications are composed by a set of related VoiceXML documents.

The following is the sequence of operations that take place when a user access a vocal service (Figure 2):

- The user dials the number of the service by using a mobile or wired phone.
- The call is intercepted and managed by a gateway that converts it into a request towards the call flow interpreter that executes the application.
- The interpreter may interact with a database where the application data is stored.
- The request is processed and, depending on the specifications of the vocal application, a response is generated and sent to the gateway.
- Once received on the gateway, the response is converted, by means of a text-to-speech (TTS) engine, into a vocal signal that is transmitted to the user.

Every time the user receives a vocal reply, he can interact with the service vocally (an ASR system converts the vocal input into program input) or by using DTMF tones. The subsequent vocal reply is generated by the application engine on the base of the user’s choice.

2.1 VoiceXML

VoiceXML is a standard XML format, promoted by W3C, that can be used to specify interactive vocal dialogs between a human and a machine [8]. VoiceXML is a declarative language: in their simplest form, VoiceXML applications describe the interaction sequence that should take place with a user, likewise a web-based experience is defined in terms of HTML pages. VoiceXML also includes support for incorporating ECMAScript code within documents. Such code can be used to dynamically change the call flow or to evaluate the input coming from the user.

To have an idea of the format of VoiceXML applications, we can report the code of the classic “Hello World” application:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xml xmlns="http://www.w3.org/2001/vxml"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.w3.org/2001/XMLSchema-instance"
     version="2.0">
  <form id="get_department">
    <field name="department">
      <grammar type="application/srgs+xml"
              src="grammars/served_departments.grxml"/>
    </field>
    <prompt>Which department?</prompt>
    <field name="department">
      <if cond="department == 'physics'">
        <prompt>Note, the department of physics has moved.</prompt>
      </if>
    </field>
  </form>
</xml>
```

The top-level element `xml` acts as a container for dialogs. Two types of dialogs are available: `forms`, used to present information to the user and to receive input, and `menus`, used to select a choice among a set of alternatives. In the case of the Hello World application, the `form` element contains just an item, the `block` element, that specifies the message that has to be pronounced by the system.

The following fragment of code illustrates how the input of the user can be acquired through a form:

```xml
<form id="get_department">
  <field name="department">
    <grammar type="application/srgs+xml"
            src="grammars/served_departments.grxml"/>
  </field>
  <prompt>Which department?</prompt>
  <field name="department">
    <if cond="department == 'physics'">
      <prompt>Note, the department of physics has moved.</prompt>
    </if>
  </field>
</form>
```

VoiceXML allows the server-side to receive input by means of the `field` tag. The text within the `prompt` element is used to specify what we expect from the user. A grammar is associated to the user input and is used to validate it. In this example, the type of
the grammar is application/srgs+xml and it is defined within the grammars/served_departments.grxml file. The filled tag is used to specify the actions to be carried out depending on the user’s input. Other elements can be used to define an opportune recovery strategy if the user’s input does not conform to the specified grammar.

There are several development environments that ease the design, implementation and testing of vocal applications. Among the commercially available ones we can report the following systems: Voice-Genie Developer Workshop (freely downloadable), Voxeo (web-based), and Tellme studio (a development tool based on Microsoft’s Visual Studio Domain Specific Language toolkit).

3. VISUAL ACCESS TO VOICEXML-BASED APPLICATIONS

After having illustrated the basics of the VoiceXML language, and the user-server interaction in the usual voice-based scenario, we can now describe the operations that take place in the visual version of the service.

We designed and implemented an interpreter of the VoiceXML language, called VOXI (VOice Xml Interpreter), that interacts with the server and translates into GUI elements the information received as VoiceXML documents (Figure 3). The interpreter is executed on the client side, i.e. on the mobile phone of the user. It is important to notice that VOXI is not dependent on a single application, it can instead be used for the interpretation of any vocal application that is installed on the server as long as it is described by sequences of syntactically conforming documents. However the interpreter is itself dependent on the hosting platform: for this reason we implemented two versions of VOXI, one for the Java Micro Edition (JME) platform, and the other for the Android platform.

VOXI establishes a HTTP connection with the server, and sends, through such connection, a request that specifies its needs. The server, analogously to the vocal case, generates an opportune VoiceXML document and transmits it to VOXI. The latter, once the document has been received, parses the VoiceXML tags and maps them onto the graphical elements provided by the underlying hw/sw platform. Then, the resulting GUI is displayed to the user.

For instance, let us consider the following fragment of a VoiceXML document:

<form id="welcome">
  <block>
    Hello!
    This is a demonstration of the service...
  </block>
  <prompt>goto next="#main"/></prompt>
</form>

The previous document is rendered by VOXI as described in Figure 4(b). Some elements and attributes are ignored in the visual representation of VoiceXML documents.
3.1 Architecture of the interpreter

3.1.1 GUI rendering
VOXI includes a XML parser that is used to validate the document and to extract its content. In particular, VOXI makes use of a SAX parser that scans sequentially the document and generates opportune events when an element starts or finishes, and when there is some content within an element. Such events are handled by VOXI that creates and fills an internal data structure that represents the VoiceXML document. This data structure is organized as a tree where the nodes correspond to containers and visual blocks of the GUI.

Once the tree data structure has been generated, VOXI renders the GUI. To this purpose, we adopted a well-known technique that is frequently adopted for the management of graphical interfaces: every element print its representation onto the display, and asks its children to print themselves. Every node of the tree data structure belongs to a given type that corresponds to the VoiceXML element from which it has been generated (Figure 5(a)). Every node type re-defines a print() method that, after having filled the part of the interface that belongs to the node, invokes the same method on all the child nodes (this is done through a printChildren() method that is inherited from the base class of all node types). Thus, it is sufficient to call the print() method on the root of the tree to obtain the whole GUI (Figure 5(b)). Some nodes do not have a concrete visual representation on the screen, but only act as containers for the child nodes.

3.1.2 Evaluation of expressions
VoiceXML documents may include ECMAScript expressions. More in detail, such expressions can be found i) in the cond attribute of the tags if, else, and block; ii) in the expr attribute of the tags var, assign; iii) in the expr attribute of the goto tag. In the first case, the ECMAScript expression, that returns a boolean value, can be used to decide the flow of execution. In the second case, expressions allow the programmer to initialize or to assign a value to variables declared within the VoiceXML document. In the third case, expressions can be used to generate dynamic URLs from which the subsequent VoiceXML documents must be retrieved.

To handle these needs, we equipped VOXI with a component that is able to evaluate ECMAScript expressions. The component interacts with a freely available and minimal ECMAScript interpreter, MiniJoe, that can be executed within the JME and Android environments. A context, associated to the VoiceXML document, is used to bind variable names to their values. VOXI extracts the expressions from the VoiceXML document and passes them to the interpreter which, by accessing the context, evaluates them. The final result is then returned to the main VOXI components that uses it for controlling the flow etc.

3.1.3 Grammars
The field element is used to request an input from the user. Usually, a grammar is associated to such element in order to verify the correctness of the input, for example to check that a value is expressed through a given number of digits.

The programmer attaches syntactical rules to the document elements by using the grammar tag. Rules can be "inline" or "external". In the first case the rule is directly expressed within the grammar tag. In the second case the grammar tag contains a src attribute that specifies the URL of the external document that contains the rules.

VOXI implements the factory pattern, so that different grammars can be handled and new ones can be easily incorporated.

3.2 Linguistic issues
The implementation techniques described above give rise, in few cases, to some linguistic inconsistencies, due to the different presentation needs of visual and vocal modalities.

Let us suppose, for example, that the user is asked to choose his favorite musical band among a set of predefined choices (such as "Beatles", "U2", "Rolling Stones", and "Pink Floyd"). The selection of the user could be done through the ASR component that recognizes the option pronounced by the user. VOXI represents the VoiceXML document as a screen containing a set of four radio buttons, where each item corresponds to one of the possible choices, and the selection is performed by selecting one of the buttons through the keyboard or the touch screen.

The problem comes from the content of the message that precedes the four possible options: in fact, the message is often something like "Pronounce the name of your favorite band". But the term "pronounce" does not make any sense in the visual representation that is displayed on the screen. This kind of problems become manifest each time the content of messages assumes that the interaction between the user and the machine is vocal.

Considering that in many cases vocal services are internationalized, i.e. they are available in different languages, we solved the problem by defining specific languages that must be used when the service is accessed through VOXI. If a service is available in English and Italian, the service provider must translate it also in "English_visual", and "Italian_visual". In "English_visual" messages will use the term "select" instead of "pronounce", and so on. It should be noted that this is not a major effort for the service provider, since the development environments adopted for building vocal applications usually include tools to ease the internationalization of the service. Moreover, such operation has to be carried out only if the service includes linguistic inconsistencies, and therefore it is not always required. When VOXI sends the first HTTP request to the server, the latter can use a parameter included within the request to automatically select the proper language.

4. RELATED WORK
VoiceXML is part of the speech interface languages (SILs) defined by the World Wide Web Consortium (W3C). The standard includes: VoiceXML, used to control the interaction flow between the user and the machine; Speech Sintesyx Markup Language, used for assisting the generation of synthetic speech (to control aspects of speech output such as pronunciation, volume, pitch and rate); Speech Recognition Grammar Specification, to permit a speech application to indicate to a recognizer what it should listen for (words and patterns); Semantic Interpretation for Speech Recognition, used for transforming, into an easily processable representation, the information acquired by means of speech recognition; Call Control XML, provides means for describing advanced telephony call control (for example multi-party calls); Pronunciation Lexicons Specification, useful to describe the way words should be
pronounced. An introduction to VoiceXML and to the other SIL standards can be found in [2].

Previous work that is related to our system belongs to the following two research fields: multi-modal interaction and auto-generation of user interfaces. With multi-modal interaction the user is able to give inputs in various modes, such as keyboard, mouse, speech, touch, and manual gestures. Similarly, the output to the user is delivered by using both visual and vocal modalities. The reason that pushes towards multi-modal interfaces is increased usability: depending on the user context and adopted device a modality can be favorable with respect to another one, and vice versa. With auto-generation of user interfaces the programmer is relieved from the burden of producing different interfaces for a single application. Instead, proper languages and/or translation mechanisms generate the user interface for web browsers, voice browsers, mobile phones, etc.

In [1], the authors studied different architectures for multimodal browsing based on XML and XSL. A first solution associates a VoiceXML document to every HTML document that has to be rendered as multimodal. The client caches locally the HTML pages and renders them through a standard web browser. The pages are also parsed by a HTML-VoiceXML converter that extracts the information needed for the dialog between the human and the machine. The VoiceXML documents are sent to a voice browser, executed on the client, that manages the interaction with the user through the TTS and ASR engines. A major problem of multi-modal interaction using two browser is synchronization: vocal and graphical content that is delivered to the user must be synchronized, and the same applies for input that can be performed, at the same time, through voice and keyboard. Another possibility is to translate XML documents into VoiceXML documents through XSL. This possibility reduces the problems due to synchronization, but requires that the web pages are written in XML.

The use of UIML for the description of generic interfaces is discussed in [4], where the authors describe a visual programming environment, with drag and drop facilities, that can be used to create different version of the same interface, depending on the target device. The high level description of the user interface, based on UIML, can be translated into VoiceXML, Java Micro Edition (JME), HTML, and WML. The authors identified a set of generic widget (Frame, MenuItem, TextEntry, etc.) and established a mapping to specific elements of the target language. The authors also highlight the difference between voice-based and graphic-based user interfaces: the ephemeral nature of voice forces the interaction between the user and the machine into a serial turn-based model, while the persistence of information displayed by graphical user interfaces suggests a more parallel style of interaction.

Another architecture that provides multimodal interfaces on mobile phones is described in [5]. The paper proposes a server side solution, i.e. the TTS and ASR components are located on the server and not on the client device. The multimodal server is composed of three elements: a voice manager, that receives the data coming from the microphone, a visual manager, that controls the graphical output on the mobile device, and a dialog manager, that controls the interaction flow.

The authors of [7] present an architecture for the automatic generation of multimodal user interfaces for Web services. The system takes the WSDL description of a service as input, and produces an interface based on XForms. VoiceXML dialogs can be embedded into the XForm document to enable a vocal interaction with the offered service. The generation of user interface does not take place on the client device, it is instead operated on a server machine where the service is published.

Other similar works include [6, 3]. It is worthwhile to notice that the motivation behind the systems described above is somehow opposite to the one behind our system. While the other systems try to extend with voice existing applications, VOXI, on the contrary, provides a GUI for existing vocal applications. Moreover, as far as we know, VOXI is the first system where the conversion between voice-based and GUI-based interaction takes place directly on the mobile device. The rationale that motivates this choice lies in the fact that, in this way, it is possible to generate a graphical user interface that is peculiar of the specific device where it has to be visualized. This feature is in large part automatically achieved by the adoption of the JME and Android platforms, which take care of displaying the widgets according to the characteristics of the hosting device.
5. CONCLUSION
In this paper, we have described an architecture that enables mobile visual access to existing vocal applications. VOXI, our VoiceXML interpreter, transforms vocal dialogs into a sequence of screens shown to the user, removing some usability and accessibility problems that are typical of voice-based systems.

The effort required to convert a vocal application so that it can be accessed by VOXI is very limited and it can be compared to the task of application internationalization and localization. Thus, no architectural changes are necessary for both applications and systems.

VoiceXML documents are, in general, small in size. As a consequence, the amount of traffic generated by the mobile device is acceptable, and the time needed to retrieve a new document does not reduce the usability of the interface. In case of large documents, the interpreter could include a prefetching unit, in charge of retrieving the VoiceXML documents before they are needed, and caching them for subsequent use.

6. ACKNOWLEDGMENTS
We would like to thank E. Coca, F. Formosa, and Liberologico srl for their contribution.

7. REFERENCES