

Interaction Design Issues for Vocal User Interfaces

J. R. Fisher
jrfisher@dtail.net

Abstract. This note discusses the design of an experimental API (application program interface) for *vocal user interfaces*. The emphasis is on *component design* which supports *vocal interaction design issues*. Related matters, including *interaction diagrams*, voice XML and intelligent AI techniques are also briefly discussed. **Keywords:** *vocal user interfaces, vocal interaction design.*

1. Background information and demo

A *voice recognition engine* (VRE) provides mechanisms for converting vocalizations into text. Commercial products like *Dragon Naturally Speaking* and *Via Voice* are readily available. The key feature of existing VREs is their ability to reliably produce speech in the form of text that can be directed into any conventional graphical text component. Say something to your favorite VRE and it will generate keyboard characters that stream into the user's selected and focused text component based upon the VRE's understanding of the vocal inputs.

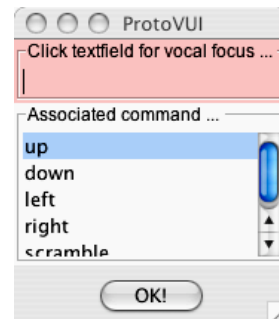
A *vocal user interface* (VUI) provides mechanisms for controlling software with vocal interactions once the speech has been converted to text. Using a VUI, we can intercept special commands that can be assigned specific meanings, and elicit certain responses in the software. For a simple *motivational* example, see reference [1]. That example illustrates a Java (applet) program, which presents the user with an 8-puzzle picture to unscramble ...



When the expert user unscrambles the puzzle, the picture image is revealed ...



The user may *interact* with the puzzle by clicking on the tiles (which are apparent in the first image) to move them, or the user may elicit a VUI to talk to. The graphical VUI artifact looks like this ...

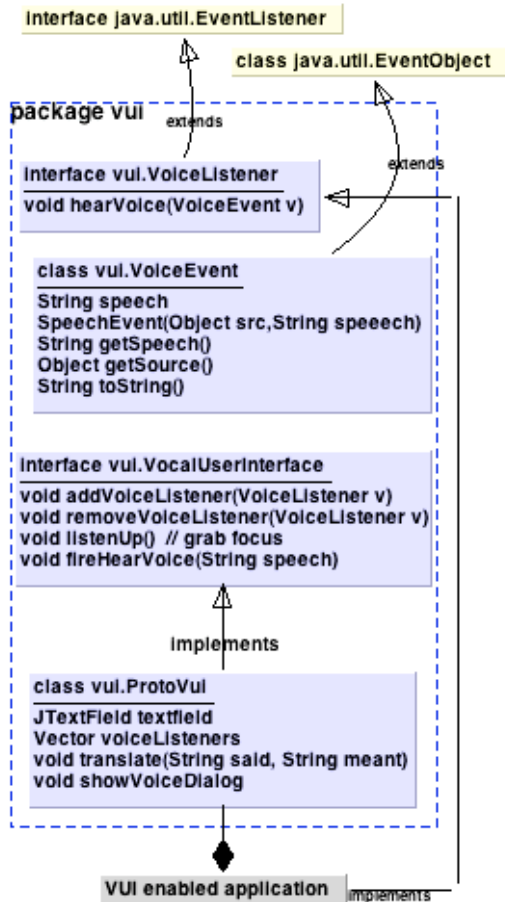


The customized VUI allows direct interaction between the user's voice and the software's response in the following way: The user talks to the VUI and the recognized speech appears in the text field. If the utterance is understood then the corresponding software response is *activated* or performed automatically. If the utterance was misunderstood, the skewed text appears in the text field and the user may select the associated command from the text area and click OK, and then the corresponding software response is performed and remembered for the next time that that specific skewed utterance is given.

In an ideal situation, "What you see is what you get" (WYSIWYG) would be extended to mean, "What you hear is what you get" (WYHIWYG). That is, if software VUI interface hears meaning M and M is understood, then M *happens*.

2. A Java VUI package

These notes describe some general capabilities for combining the abilities of VREs and custom VUIs. The following diagram shows the classes in the vui package.



The diagram is consistent with the following general design intentions for a VUI:

A *VocalUserInterface* (such as the ProtoVUI class) should

- Be able to capture speech as text.
- Maintain a list of *VoiceListeners* by adding new ones or removing old ones.
- Be able to notify *VoiceListeners* of a *VoiceEvent* directed at the *VocalUserInterface*.
- Support being told to focus on speech capture by any one of the *VoiceListeners*.

A *VoiceListener* (such as the 8Puzzle sample program) should

- Be able to hear a *VoiceEvent* and
- Act accordingly, depending on the event (who said, what said).

A *VoiceEvent* should

- Encapsulate text speech (what said).
- Identify the source (who said).

3. Using the vui package

A voice activator can be any class that implements the *VoiceListener* interface. In the 8Puzzle example we decouple the voice activator from the other components by defining a separate activator, as shown in the following code display

```

import vui.* ;

/**
 * Activator class for voice commands.
 */
class PuzzleVoiceActivator implements
    VoiceListener {
    Image8Puzzle puzzle ;

    public PuzzleVoiceActivator(Image8Puzzle p) {
        this.puzzle = p ;
    }

    /**
     * Notice clever way that
     * focus is thrown back to the VUI
     * that fired the VoiceEvent.
     */
    public void hearVoice(VoiceEvent v) {
        String cmd = v.getSpeech() ; // what was said
        if (cmd.equals("up")) puzzle.up() ;
        else if (cmd.equals("down")) puzzle.down() ;
        else if (cmd.equals("left")) puzzle.left() ;
        else if (cmd.equals("right")) puzzle.right() ;
        else if (cmd.equals("scramble"))
            puzzle.scramble() ;
        // FORCE FOCUS BACK TO SOURCE
        ((ProtoVUI)v.getSource()).listenUp() ;
    }
}
  
```

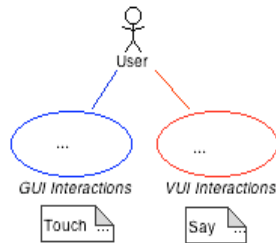
An application creates its *VocalUserInterface* and establishes basic speech text elements. For example, in the constructor for the Image8Puzzle panel

```

...
voiceActivator = new PuzzleVoiceActivator(this) ;
// vui
vui = new ProtoVUI() ;
vui.translate("move tile up","up") ;
vui.translate("up","up") ;
vui.translate("move tile down","down") ;
vui.translate("down","down") ;
vui.translate("move tile left","left") ;
vui.translate("left","left") ;
vui.translate("move tile right","right") ;
vui.translate("right","right") ;
vui.translate("scramble puzzle","scramble") ;
vui.translate("scramble","scramble") ;
vui.addVoiceListener(this.voiceActivator) ;
...
  
```

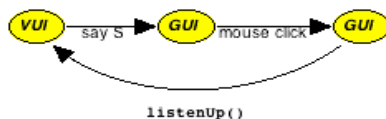
4. Interaction diagrams for VUIs and GUIs

Generally speaking, user interactions with software can be gui-enabled or vui-enabled or both. To be gui-enabled means that keyboard, mouse, joystick, etc. are active (or could be). To be vui-enabled means that there is some mechanism (as suggested above...) for making a VRE active by means of a voice activator. There are many interesting interaction possibilities for programs that are both gui-enabled and vui-enabled. It should be possible to combine vocalization or vocal gesture (VUI) with graphical gesture (GUI).



At present, it is not clear whether these aspects should be necessarily separated or combined. Separating these aspects seems to reflect the current preference for GUIs: VUIs are extremely underutilized. But conceptually there is a strong impetus for combining VUIs and GUIs, because that corresponds to the normal human communication modality of speech and gesture.

For example, consider a drawing application that allows many different kinds of shapes to be drawn at various locations on the drawing panel. A possible interaction would be to allow the user to say what kind of shape is wanted and then to click the mouse on the spot to draw it, as depicted in the following diagram



The diagram suggests that the normal interaction state is for the VUI to be focused. The user then says something S, followed by a mouse click (which will cause focus in the GUI context), and then the activator calls `listenUp` (a VUI message) to cause focus to return to the VUI.

Interaction state diagrams like the one above can be helpful in the interaction design of systems with multiple user interfaces (VUIs + GUIs). Even better would be the development of good tools for automatically generating control code for applications based upon the diagrams. Using *VoiceEvents* that encapsulate both speech and

originating source provides a mechanism (*who* said *what*) to avoid having to use a more elaborate focus manager. Still, it seems that the *focus problem* itself is not insignificant and not well understood at this time!

5. Related issues and conclusion

Voice recognition agility is currently robust and promising. (Improvements will always come.) Voice XML (vXML) offers promising capabilities for vocal interaction with web pages and for information retrieval by means of spoken input. However, this paper is mostly concerned with the component design and interaction design of programs that include vocal inputs. For example, we discussed the important and difficult focus problem in the last section. Several of these interaction issues are unlikely to be solved using *only* the current *data* capabilities of vXML.

Another issue is to incorporate intelligent interaction using AI techniques of language understanding so that more sophisticated natural language can be used to interact with programs.

It seems clear that program component design, interaction design, vXML data transfer and AI techniques need to be combined into a larger set of vocal solutions for software design that afford realistic human vocal interaction with computers.

In conclusion, the API presented here does seem to allow for *partially adequate* vocal user interface implementations, provided that user interactions are carefully analyzed and specified. The separation of voice events from other interaction events seems like a valuable approach. More elaborate and natural interaction designs must wait for *better analysis of the relevant interactions*.

6. References

- [1] Interactive VUI demo at <http://dtail.net/vui>
- [2] Allen, J., *Natural Language Understanding*, Benjamin/Cummings, 1988.
- [3] Preece, J., Rogers, Y., Sharp, H., *Interaction Design*, Wiley, 2002.
- [4] *W3C vXML current*: <http://www.w3.org/TR/voicexml20/>