Programming by Voice: A Hands-Free Approach for Motorically Challenged Children

Amber Wagner
Dept. of Computer Science
University of Alabama
ankrug@bama.ua.edu

Ramaraju Rudraraju
CIS Department
UAB
ramaraju@uab.edu

Srinivasa Datla
CIS Department
UAB
srini.datl@gmail.com

Avishek Banerjee
CIS Department
UAB
avi1986@uab.edu

Mandar Sudame
CIS Department
UAB
mandars@uab.edu

Jeff Gray
Dept. of Computer Science
University of Alabama
gray@cs.ua.edu

Abstract

This paper introduces a voice-driven tool applied to an Initial Programming Environment (IPE), which gives motorically challenged individuals the opportunity to learn programming skills; in particular, our project allows programming by voice within Scratch. Although the native Scratch environment allows users to create a program by arranging graphical blocks logically, such visual languages are completely dependent on the use of a mouse and keyboard. This modality of interaction limits users based on physical abilities. Our solution is a tool, called Myna, which is a voice-driven Java application executed parallel to Scratch. Myna processes voice commands from the user, interprets those commands according to a pre-defined grammar, and simulates synonymous actions of a mouse and keyboard within Scratch. The resulting environment assists those with a motor disability (particularly young children) in learning the joy of programming. This extended abstract describes the motivation behind the project, a technical description of Myna, and defines the current work in progress.

Author Keywords

Scratch; Accessibility; Speech; Modeling

ACM Classification Keywords

B.4.2 Input/Output Devices; H.5.2 User Interfaces

General Terms

Human Factors; Languages; Modeling

Diagnosis	Number Impacted	Age of Onset
Spinal Cord Injury	250,000	56% occur between 16-30
Muscular Dystrophy	500-600 each year	At birth
Multiple Sclerosis	250,000-350,000	20-40
Cerebral Palsy	800,000	10,000 babies born with CP each year

Table 1. List of a few major diagnoses impacting users' abilities, particularly children or young adults.

Introduction

This extended abstract describes our efforts to use voice input to drive Initial Programming Environments (IPEs) such as Scratch [7]. Scratch is a graphical programming environment developed at MIT and used in many outreach efforts to children [1], which allows users to create programs by dragging and dropping blocks of code. By using a graphical approach, Scratch prevents users from having to concentrate on syntax; however, the Scratch environment is based on the WIMP metaphor (Windows, Icon, Menu and Pointing device) [8]. The event-driven WIMP interfaces have been a dominant input modality for desktop applications and user interfaces for several decades. WIMP provides ease of use, but assumes dexterity of human hands to use a mouse and keyboard. The dependence on a mouse and keyboard poses limitations for those who have motor disabilities and are not able to exhibit the level of agility needed to manipulate and control such devices. This limitation unnecessarily excludes a significant portion of our population (see Table 1) from having access to many different desktop applications, including IPEs.

In this extended abstract, we describe Myna[1], which is a tool that runs parallel to Scratch and provides a voice interface that can be used to specify programming tasks. In Myna, the voice-driven interface enables a user to perform basic functionality (e.g., navigation across the Scratch menu structure, dragging programming blocks into the script editor), as well as additional macro commands that can be used to simulate all of the operations that could be performed in Scratch with a mouse and keyboard. The project has investigated several common interaction patterns that can be imitated through shortcuts to minimize the amount of speaking required (vocal strain is an important issue in voice-driven programming environments such that economy of expression is vital) [3]. This project unites ideas of human-computer interaction with Computer Science education to provide an assistive environment that allows a new audience to learn computational thinking.

[1] Myna is named after the species of birds that are well-known for their imitative skills.

The next section describes the support provided by Myna to vocally navigate through Scratch and the commands that can be used to insert blocks into a Scratch program. The overall architecture of Myna is then summarized along with a description of the technique we used to build the language grammars needed to integrate Myna with a voice recognition engine. The Current Work section describes how the project has progressed and an explanation of the next phase of the project. The paper concludes with a summary and description of remaining work.

Navigating By Voice

Wobbrock et al. [9] defined the concept of "ability-based design." Rather than focus on a user's disability, developers should focus on a user's ability. Table 1 displays statistics regarding various diagnoses impacting user abilities [6]. Any voice-driven user interface faces the challenge of mimicking the actions of keyboard input and mouse clicks without adding significant overhead. The challenge becomes even more obvious considering the fact that many applications are developed with the sole intention of being used with the WIMP metaphor. To overcome the challenges of mapping the WIMP metaphor to a voice-driven interface, we investigated and developed several forms of interaction, which are described in the following subsections.

Drag and Drop Navigation

The drag and drop navigation mimics the sequence of click-drag-release actions of a mouse. In Myna, whenever a user says "drag and drop" followed by the name of a Scratch language construct, Myna clicks on the control and drags it below the previously placed control. If the context of the command represents the first control to be dragged, Myna drops the control near the top of the scripts editor.

Additional Commands

Menu Commands: There are numerous menus within Scratch. We refer to these as static commands because their screen placement is static. The verbal commands for these controls match the verbiage on the control.

Macro Commands: Three macro commands have been added to better facilitate block placement. These commands take advantage of the transparent frames (the numeric identifiers in Figure 1). The existing macro commands are as follows:

1. *Drop After* – Drops the selected control after the command at the user-identified location (the number based on the transparent frame).
2. *Drop Before* – Drops the selected control before the command at the user-identified location.
3. *Drop In* – Drops the selected control (such as position 5 in Figure 1) into the user-identified location.

Continuous Navigation
Continuous navigation mimics the continuous movements of the mouse cursor. The user can say "keep moving" followed by the desired direction of the cursor (e.g., "up", "down", "left" and "right"). The mouse cursor continuously moves in the desired direction until the user says the keyword "stop." In addition, the user can also say "move" followed by the direction. In this mode, the cursor moves several pixels in that direction upon each invocation of the command. Thus, Myna provides a continuous form of navigation and also the option of one-time movement. Such flexibility is useful during situations where the user needs to click on parts of the screen that are otherwise inaccessible with the drag and drop voice commands for individual controls.

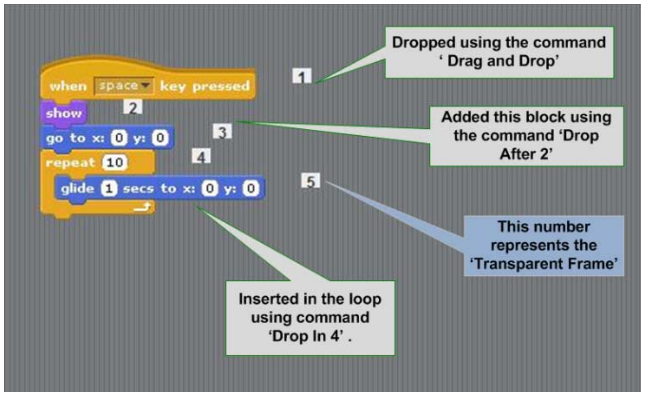
when space key pressed
show
go to x: 0 y: 0
repeat 10
glide 1 secs to x: 0 y: 0
1
2
3
4
5
Dropped using the command ' Drag and Drop'
Added this block using the command 'Drop After 2'
This number represents the 'Transparent Frame'
Inserted in the loop using command 'Drop In 4' .

Figure 1. Transparent frames added by Myna.

Navigation using transparent frames
Navigation using transparent frames makes use of small labels that mark every component in the script window (please see Figure 1). These transparent frames allow the user to say the label number in the sequence of commands in order to provide a context for movement or dropping a new programming statement. As new blocks are added to the script, Myna updates the numbers associated with each block. The sidebar on this page describes the macro commands associated with the transparent frames (line numbers) in addition to explaining the menu commands.

We created a YouTube channel that offers live video demonstrations of Myna, which is available at: `http://www.youtube.com/user/Teammyna`. Examples from the navigation commands presented in this section are available at this channel.

Myna Architecture

An important design consideration was the requirement that Myna exist outside of the Scratch implementation (i.e., Myna does not require any source code modification to the Squeak implementation of Scratch). This requirement posed several challenges (e.g., adding transparent frames as visual overlays to Scratch) that are described in this paper. The design choice to build Myna outside of the Scratch source code was made to provide flexibility for adapting Myna to future versions of Scratch as well as a mechanism to apply the Myna ideas to similar environments (e.g., Alice, AgentSheets, App Inventor, LabView, other IPEs, or even a broader range of applications). Myna is implemented as a Java application using a voice recognition front-end to drive a back-end that provides programmatic control over the mouse and keyboard (through use of the Java Robot class).

Another design goal was to use an architecture that is robust enough to accommodate future changes in Scratch's user interface (e.g., if future versions of Scratch move menu items and widgets to different parts of the Scratch screen). To assist in separating the dependencies between the user interface and the core functionality, we applied the Model View Controller (MVC) [5] design pattern, which is commonly used to maintain large and complex data sets involved in developing user interfaces. In the following subsections, we define a 'component' as a clickable area within the Scratch window, which generates an action in Scratch.

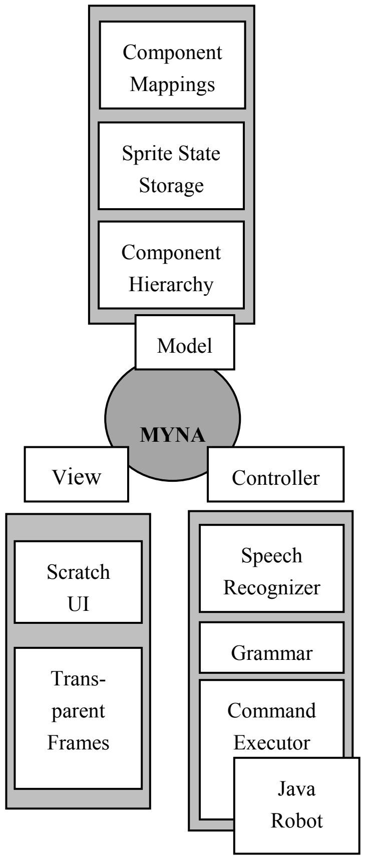

Figure 2. Myna Architecture.

Model
The model represents the category, state and low-level behavior of the component blocks, as illustrated in Figure 2.

COMPONENT MAPPINGS
The initial mapping-location on the screen of each component is recorded in a configuration file. Component Mappings is the collection of all property files used by Myna.

COMPONENT HIERARCHY
Based on the type of action generated in the Scratch window, each component is categorized into the following hierarchy:

- *Static Scratch Component*: The components that do not move from their current location are categorized as Static Components (e.g., File, Edit, and other menu commands).
- *Movable Blocks*: The components that can be dragged from their current location are categorized as moving blocks (e.g., "when clicked", "wait seconds").
- *Movable Block Containers*: Components that can contain Movable Blocks are categorized as Movable Block Containers (e.g., if, repeat, forever).
- *Sprite State Storage*: The component collection of each Sprite is stored in the Sprite State Storage. Each Sprite (i.e., each character in the Scratch window) has a collection of components currently in the script editor area of the Scratch window. Whenever a new Sprite is created or the context is switched to a different Sprite, the component collection of the currently active sprite is loaded.

Controller
Processing a user's voice command generates a request for the model to change its state and perform the respective action on the Scratch window. This is handled by the Controller, which has the following components:

SPEECH RECOGNIZER
The speech recognizer is used to recognize the voice commands as spoken by the user. Cloud Garden, which is a third-party implementation of JSAPI (Java Speech API), is used in Myna to enable voice recognition.

GRAMMAR
The Myna grammar is a collection of pre-defined words that are relevant to various activities within Scratch. The Speech Recognizer recognizes a voice command only if it is defined in the grammar file(s). A separate sidebar on the next page describes the dynamic grammars utilized in Myna.

COMMAND EXECUTOR
This component executes the input commands by performing the appropriate mouse and keyboard actions by calling methods in the Java Robot class.

JAVA ROBOT
This class contains methods to generate native system mouse and keyboard events [4]. For example, there are methods to move the mouse to a specific coordinate on the screen, to double click a mouse button, and the provide input as if it was coming from the keyboard. The name "robot" for this class is somewhat of a misnomer - the class is not focused on robotics, per se, but rather automation of mouse and keyboard input.

View
The view is a representation of the model that is visible to the user. The Scratch window itself falls under the context of the View, although no changes have been made to the source code of Scratch.

Dynamic Grammars

An important feature implemented in Myna is that any speech recognizer requires a grammar that is used to process the speech input. The grammar(s) could be classified into two types: Dictionary Grammars and Rule Grammars. They both differ in the way the pattern of words is defined and also the way they are used: a Dictionary grammar is built into the recognizer, whereas a Rule grammar is built into the application.

For any speech application, multiple grammars may be instantiated at any time. We instantiated grammar files demarcated by the context and processed on demand. This supported a lazy loading of the grammars within the recognizer, which improved accuracy and eliminated the case of improper execution of command sequences.

During execution, Myna only has three grammars active in the initial stage. They are the grammar file with the default commands (file, edit, control), the grammar file with navigation commands, and the grammar file with the speech pause and resume commands.

Myna Workflow

In this section, we demonstrate the flow of all parts of the architecture by considering a typical workflow of Myna. The following voice commands are mapped to the flow illustrated on Figure 3.

1. User says a voice command.
2. The input command is identified by the speech recognizer and checked against the grammar.
3. If the command is present in the grammar, an appropriate action is invoked in the Command Executor.
4. The Command Executor obtains the current mappings of the component.
5. If necessary, the Command Executor requests the Model to change its current state.
6. The Command executor calls into the Java Robot to perform the corresponding mouse/keyboard action.

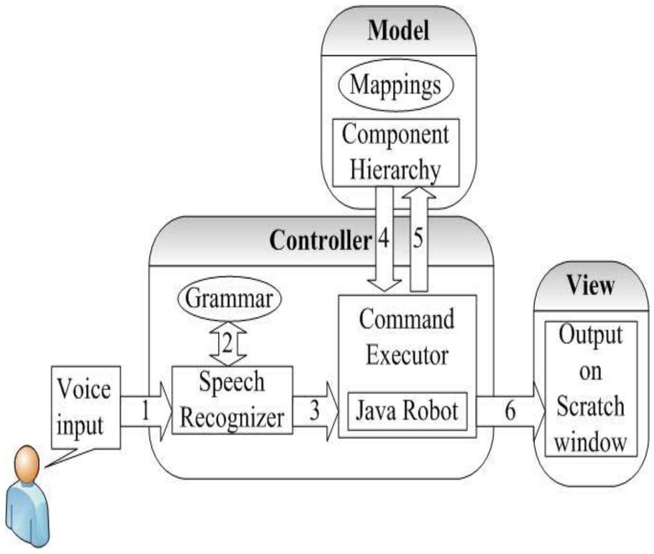

Figure 3. Myna Workflow.

Current Work

The work described in previous sections represents an existing prototype (Phase I); the final three phases are described as follows:

Phase II: Complete Myna Implementation

During this phase, the primary goal is to create a 100% voice-controlled application in order to present it to users for testing. Our requirements for meeting this goal are the following:

1. Add all static commands to the grammar.
2. Add commands for undo/redo and remove.
3. Add scroll bar navigation.
4. Create a wizard to allow for grammar customization. Harada et al. [3] demonstrated the use of syllables rather than a grammar, which may be more appropriate for individuals with Cerebral Palsy.
5. Add multi-lingual support (the wizard may solve this issue).
6. Adapt for any screen resolution.

Items 1) and 2) have been completed. Items 3), 4), and 5) will be completed as a next step. Item 6) will also be completed soon as future work. We have tested the existing application successfully on multiple machines with varying resolution.

Phase III: Myna Evaluation

Our future work is to obtain empirical data regarding the usability and performance of the application. We will perform a small user study with peers to test the application and to ensure all bugs have been removed. We also want to ensure the voice commands are intuitive.

We have been working with United Cerebral Palsy of Birmingham. They have agreed to work with us by asking their patients to evaluate the application. While the application is evaluated, the participants will be learning Computer Science skills and foundational concepts. The testing will consist of four three-hour weekend sessions involving 6-7 children. A therapist (with whom the student is familiar) and an instructor will be present for each student.

Phase IV: Myna Expansion
If the evaluation of Myna is successful, we want to expand the idea of voice-controlled programming to other IPEs (Lego Mindstorms's Labview, App Inventor, Alice, AgentSheets, etc.), or even other types of general applications. Rather than hardcode a grammar as we have done with Scratch, we will use a model-driven engineering (MDE) technique [2] to programmatically create the necessary grammar and programmatic mouse and keyboard controls. This approach will allow the developer of an application to integrate voice controls simply by creating a model that mimics the application interface.

Conclusion

This extended abstract describes our approach to formulate voice-driven programming of Initial Programming Environments (IPEs) as an alternative to the orthodox way of using a mouse and keyboard. With the primary capabilities of our voice-driven application, Scratch can be used in a manner that overcomes the physical challenges of WIMP (window, icon, menu, pointing device) metaphor, which provides a path for those with disabilities to have access to Scratch as a learning environment.

We described the three future phases of this project, which are:

Phase II: Expand the existing implementation of Myna to create a 100% voice-controlled application.

Phase III: Evaluate Myna with a small group of peers to ensure the application is usable in order to minimize frustration from our target audience. We will then work with UCP of Birmingham to evaluate Myna.

Phase IV: Using MDE, we will create a model to simplify the process of incorporating voice controls into applications built with the WIMP metaphor as the core.

Acknowledgement

This project is supported by a Google Research Award and NSF grant IIS-1117940. We thank Dr. Gary Edwards of United Cerebral Palsy of Birmingham for comments on earlier versions of this work.

References

[1]*http://info.scratch.mit.edu/About_Scratch*

[2]Gray, J., Lin, Y., and Zhang, J. Automating change evolution in model-driven engineering. In *IEEE Computer*, vol. 39, issue 2, (2006).

[3]Harada, S., Wobbrock, J., Malkin, J., Bilmes, J., and Landay, J. Longitudinal study of people learning to use continuous voice-based cursor control. In *Proc. Intl. Conf. on Human Factors in Computing Systems*, Boston 2009, (2009), 347-356.

[4]Java Robot Class: *http://forum.codecall.net/java-tutorials/25923-robot-class.html*

[5]Leff, A. and Rayfield, J.T. Web-application development using the model/view/controller design pattern. EDOC 2001: 118-127.

[6]National Institute of Neurological Disorders and Stroke. National Institutes of Health. *http://www.ninds.nih.gov/disorders/disorder_index.htm.*

[7]*Scratch. http://scratch.mit.edu.*

[8]WIMP Interfaces, Ashley George Taylor, Winter '97 *http://www.cc.gatech.edu/classes/cs6751_97_winter/Topics/dialog-wimp/*

[9]Wobbrock, J., Kane, S., Gajos, K., Harada, S., and Froehlich, J. Ability-based design: Concept, principles, and examples. In *ACM Trans. On Accessible Computing*, vol. 3, no. 3, article 9, (2011).