APL: Audio Programming Language for Blind Learners

Jaime Sánchez and Fernando Aguayo

Department of Computer Science, University of Chile
Blanco Encalada 2120, Santiago, Chile
{jsanchez, faguayo}@dcc.uchile.cl

Abstract. Programming skills are strongly emphasized in computer science. Programming languages are constructed based on sighted people as end-users. We have designed Audio Programming Language for blind learners based on audio interfaces to support novice blind learners to develop and exercise problem solving skills. APL was designed with blind learners from the beginning to construct programs and solve problems with increasingly complexity. Audio Programming Language was usability tested during and after implementation. Blind learners used, wrote programs, and helped to make improvements to this programming language. Testing results evidence that APL mapped the mental models of blind learners and helped to motivate them to write programs and thus entering to the programming field.

1 Introduction

Diverse attempts have been made to make programming closer to end-users: Basic, Logo, Smalltalk, Pascal, Boxer, Playground, KidSim, AgenSheets, LiveWord, Shoptalk. All of these programming languages have contributed to expand the number of people who can program. Many of them have applied user interfaces principles to programming. This has ended up with better skills for novice learners to program. Thus the literature describes studies concerning programming by demonstration, programming by example, visual programming, graphical programming, and physical programming [2,3,5,7]. Most of these programming attempts (if not all) have been focused on visual programmers.

To write a program using these languages learners with visual disabilities has to use text-to-speech systems that "read" programming commands and variables assuming that they follow easily the same logic of programming used by sighted learners.

Recent studies have shown that by using audio-based applications blind children can develop and rehearse cognition [1,4,6,8,9,10]. Most of these studies focus on the development of 3D audio interfaces to map the entire surrounding space and thus helping blind children to construct cognition through audio-based interfaces such as tempo-spatial relationships, short-term memory, abstract memory, spatial abstraction, haptic perception, and mathematic reasoning.

This research study introduces APL, Audio Programming Language. APL is based on audio to enhance problem solving and thinking skills in novice blind learners. APL is not thought as an alternative to conventional programming languages, rather it is a tool to help to better integrate novice programmers to the field of conventional languages. In doing so APL can generate Java code and thus make that novice programs can be used by other people without needing APL. They will just need Java and a TTS engine. APL is a tool to motivate blind learners to enter to the programming world.

2 Programming by Blind Learners

What are the specific needs of blind learners to program? Why is it difficult for novice blind learners to map and follow current programming languages? Can we develop a language to fit the needs of these learners, especially those that will not follow a computer science career? These were some of the underlying questions in our study.

A programming language communicates the programmer with the computer. To do this the structure and logic are designed in such a way to be interpreted by the machine. Actual languages are based on the idea that a programmer writes command lines interpreted by the computer. These commands must be correctly written and well defined otherwise the machine cannot understand instructions and specific tasks are unsolved. This implies to memorize a huge amount of command lines and to write them correctly in order to avoid error parsing. Most of these programming languages are heavily based on visual interfaces.

Two major difficulties can be found when blind learners use these languages. First, if they use a pure language they face the issue of verifying the program consistency and the correct reading of command lines. Second, if we provide them current tools to support program construction, they deal with graphical user interfaces. APL intends to close the gap between programming by sighted learners and programming by blind learners.

Fig. 1. A blind learner programming with APL

APL is the first intent to develop a programming language oriented to blind learners. They can interact and communicate with the computer as sighted programmers do by writing their own programs and exchanging them with other users, sighted or blind. The only difference is the way they do it and the sensory channel used to interact with the machine.

To attain our goals APL was designed to make easier the correct writing of command lines and to avoid parsing problems. We facilitate machine-programmer interaction through additional tools not commonly found in current programming languages.

Common programming languages use minimum storage units or variables that can be appropriate to the user's needs. It is hard to imagine a language without string or integer variables. In contrast, APL implements a new and unconventional type of variable to store sounds to be later manipulated by blind users. Sound is a fundamental unit of APL.

3 Audio Programming Language

We aimed at developing a tool to facilitate the manipulation of a programming language by blind learners. APL is oriented to blind novice programmers by diminishing syntax complexity of a current language and adding functions to analyze the final program. Learners can get programs in Java code with the help of an APL code generator that makes the necessary conversions to add the required functionalities.

3.1 Model

We developed an implementation model for APL. Two basic conditions were set: The software should be easy to implement and flexible by allowing small changes without having to modify the whole program. APL has three main layers: Audio interface, programming interface and programming logic (see Figure 2).

Audio Interface includes two states: TTS and Recorded Text. TTS is used to reproduce words and sentences. Recorded Text is used to give command feedback when running the round list and to pronounce letters and numbers.

Programming interface contains three states: Round list, keyboard, and Menu. Round list includes all commands used in the program since the list consistency is operated dynamically by the command integrity. Keyboard, which can be activated any time the novice programmer enters information to the program such selecting a command from the round list, variable names, and text for output. Menu is active during the whole use of APL and allows executing the program or exporting this to Java code

Programming Logic includes four states: Command integrity, free input, run program, and export to Java. Command integrity is in charge of maintaining the integrity of the program by including or eliminating commands in the round list to help programmers to use adequate commands and communicating with the TTS or recorded text. Free input allows users to enter text freely providing permanent feedback through the integration of the TTS (words and sentence) and the recorder text (letters and numbers). Run program executes the program if the command state is correct. Export to Java exports the program through a code generator.

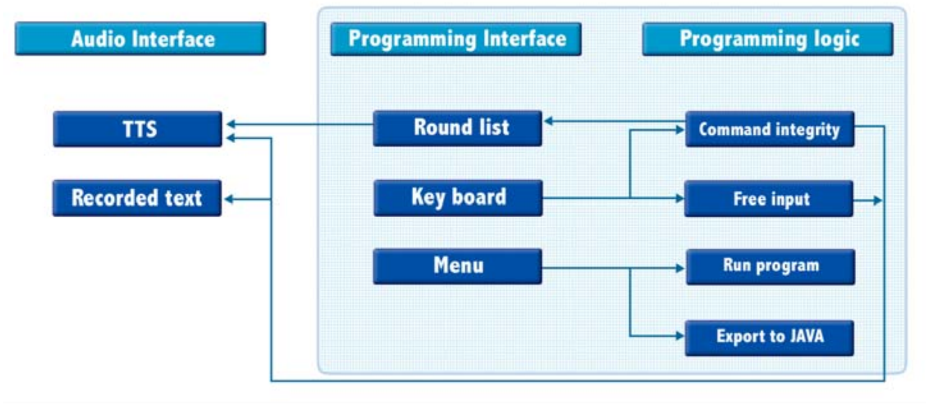

Fig. 2. Layers of APL

3.2 Implementation

APL was developed by using Java 1.4.1 and FreeTTS Java speech synthesizer. These are fine tools for media management and close to the machine logic. APL has the following modules: DataBase, Integrity, Kernel, CHI and generador de JAVA.

DataBase stores the commands and its integrity. Integrity reads the DataBase and loads a certain structure and commands. Kernel manages the interaction between the user and programming rules. CHI controls input/output audio and translates input/output text to audio. The interface is entirely based on audio in such a way that blind programmers have not direct interaction with the screen. APL has two modes: programmer mode and running mode.

There is no programming language skills prerequisite to use APL. Learners don't need to memorize any commands. APL has a circular command list introduced to the user to select the desired command. The list has options related to the actual state of the system. In doing so two current issues in conventional programming can be solved: command navigation and command semantic.

To control command navigation APL presents a reduced command list to improve the navigation through the list and to optimize the programming time. Command semantic is prevented by assuring that the selected command is correctly written in such a way that APL can interpret it in the runtime.

The circular list of APL is dynamic. The main list consists of input, output, cycle, condition, and variable (see Figure 3). Each of these commands possesses an integrity table placed on the programming logic layer (see Figure 2) to upgrade the circular list with new commands. APL can request answers to a set of questions to complete the command line at the audio interface layer. This process can be repeated as many times as necessary to write a program. The command definition is:

Variable: It consists of variable definitions. They can be text, numeric or sound. Text is read through text-to-speech.

Input: The blind programmer uses this command to create an input requirement when executing the program by selecting the desired variable, keyboard or sound. This input should be saved as a variable defined by the programmer.

Output: It is the medium available to the programmer to create an output of the product. Here APL uses only sound interfaces. Sound has two modes: 1. The programmer defines a variable and saves it with a corresponding sound, and 2. The programmer defines a variable and saves it with a corresponding word or sentence entered through the keyboard and read as text-to-speech when executing the program.

Condition: It is a command that compares variables defined by the programmer. If variables are true the commands between tags are executed and a new option is created in the command list, end of condition.

Cycle: It is a command that allows executing the program many times in a defined sector. It comes out of the sector when the defined cycle condition is attained. The cycle is completed by using the end cycle option.

For feedback optimization when the programmer writes the program we used a mix solution of TTS and recorded text to provide an immediate response when writing since we used recorded letters and numbers to avoid TTS processing.

4 Usability Testing

APL was built by and for blind learners. We implemented a qualitative case study methodology by observing and recording the interaction of end-users and experts with the program. APL was usability tested during and after development.

Expert users: Immediately after APL was implemented we tested the software with three expert users that have experience and knowledge in programming to validate the functionality and to identify possible problems and errors in order to get feedback, suggestions, and make some improvements. They were informed about the correspondence of APL commands with generic commands and wrote programs using APL. They also made comments and answered pre-designed questions detecting functioning problems such as: a. APL did not support multiple cycles, b. APL did not allow to make waterfalls conditions, c. The conditions of APL did not allow to exit from cycles, d. Some issues occurred when manipulating "keyboard" variables (variable types such int, var, and string cannot be defined), e. Sound and voices can be confusing in some passages, f. The executor went down when the cycle was not appropriately defined. Then we analyzed in depth the content and comments given by experts and redesigned the prototype before the blind users tried it.

End-users: APL was evaluated through a qualitative case study methodology by observing and recording the user's interaction with APL. The study included a two stages usability pilot testing with three blind novice programming learners, ages 17 to 20. During the first stage learners followed four interactive sessions of 2.5 hours each. First, basic programming concepts were introduced to them and after that they solved related mental tasks. Second, they interacted with APL by using basic commands to create simple programs such as Hello World and guessing games. Third, they wrote their own programs such as “think an idea and make a program”. Finally, they wrote a small program for blind and sighted people to learn Braille (see Figure 1).

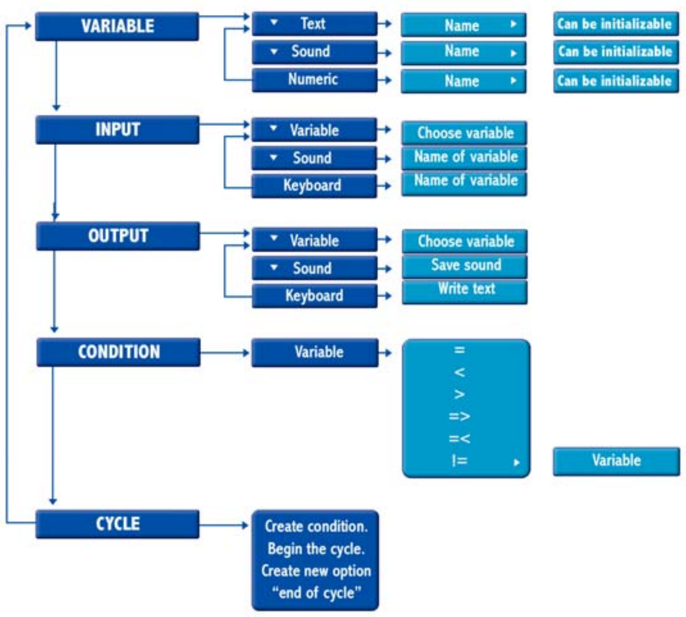

Fig. 3. Command list tree

The main goal was to familiarize users with concepts and elements that can be used to solve problems with APL. The idea was having users to interact with APL, to identify classic algorithms and data structure in computer science, and to develop algorithmic thinking to solve a problem. Users understood the functioning and the main goal of APL. They also learned about the basics of programming.

The second stage consisted of answering a usability evaluation test and having an interview. The test consisted of eighteen user satisfaction questions in a Likert type scale. The interview pursued to get in depth answers about APL and to record comments about their experience. Blind users’ behaviors were observed by a team of three researchers.

In the beginning learners understood some programming concepts but it was complex to them to understand the concept and functions of different commands such as cycle, condition, and variable. This understanding took more time as expected. During the third session they understood these concepts and made comments concerning APL based on their interaction experience and expectations about the functionality. We observed that methodologies such as theoretical explanations and step by step exercises did not help them to understand fully the basic concepts of programming. This can be explained due to the fact that blind learners tend to rely heavily on concrete experience before building abstract thinking. They did not construct a mental model of APL and programming until they designed their own idea and understood the meaning fully. Then through passing from the known to the unknown they increasingly developed understanding of programming.

Blind learners were able to interact with APL by using commands such as input, output, variables, cycles, and condition. They could write some programs following step by step directions as well as writing programs emerging from their own ideas. During the last session they evidenced to understand programming concepts and apply them to their programs. In doing so blind users constructed logic thinking skills, verbalized them with their partners and observers, and wrote programs to prove their feasibility. They were very motivated and satisfied as a result of interacting with APL. We assume these are preliminary results but very promising.

As users understood concepts, used commands, and applied them to their programming they were able to design logic instructions, understand, reproduce, and verbalize them. They also constructed algorithms that could be reproduced mentally. They showed high interest and enthusiasm when programming.

Blind learners also made diverse comments and suggestions to improve APL such as to include a feedback by using the space bar to remind the last written word, to start up APL by their own, and to make it faster when using arrows.

Our experience with blind users programming APL shows that current programming visual tools are not feasible for these learners. It appears that the issue for novice learners is not just to adapt visual programming to blind learners [11]. The theoretical background of these tools is complex to be easily mapped by novice blind learners. One interesting result of our usability test evidenced that interactivity may have rather different meaning and implications to blind users.

TTS and recorded text were tested separately from the use of APL to evaluate TTS and recorded text feedback for letters and number pronunciation. Results were entirely satisfactory demonstrating the user's preference for the integrated system (TTS and recorded text) instead of the previous system with a TTS processing all letter conventions.

5 Conclusion

We have introduced APL, Audio Programming Language for blind learners. APL is designed to help novice blind learners to enter to the programming world and to solve problems and develop thinking skills by targeting their needs and mental models. Learners were able to interact and program APL by using audio-based commands. They understood programming concepts and apply them by programming their own ideas.

Adequate uses of audio as a sensory interactive medium can help learners to learn how to program. We initially observed that blind learners can develop algorithmic and logic thinking skills with APL. They verbalized these skills, and wrote programs to prove their feasibility. The experience of interacting with APL was motivating and unique to them by expressing satisfaction with the programming experience. They have continued to program with APL after participating in the study.

APL is not an alternative proposal to visual programming for blind users. We believe that for higher level tasks there should be interfaces to allow an adequate access to visual programming such as in [12]. Our proposal goes in a different direction. We aimed at motivating novice blind learners to learn programming basic ideas and solve problems with APL. Our idea is that their programs can be used by other people. To do this we have embedded in APL a Java code generator.

We are learning the way blind users map the programming process by using APL. It may be somehow different from sighted users. Interacting through programming may have a different meaning to them. This should be studied more fully in future studies. Finally, APL is a first step to provide a robust Audio Programming Language to blind learners and thus helping them to enjoy programming and developing their cognition.

Acknowledgments

This report was funded by the Chilean National Fund of Science and Technology, Fondecyt, Project 1030158.

References

1. Baldis, J. Effects of spatial audio on memory, comprehension, and preference during desktop conferences. Proceeding of the ACM CHI ´01, Vol 3, 1, (2001), pp. 166-173
2. Cypher, A., Halbert, D. C., Kurlander, D.: Lieberman, H., Maulsby, D., Mayers, B.A., and Turransky, A. (eds.). Watch What I Do: Programming by Demonstration. Cambridge, MA: The MIT Press, 1993.
3. Lieberman, H. Your Wish Is My Command: Programming by Example. Morgan Kaufmann, San Francisco, 2001.
4. McCrindle, R. & Symons, D. Audio space invaders. Proceedings of the Third International Conference on Disability, Virtual Reality and Associated Technologies, (2000), pp. 59-65.
5. McDaniel, R. and Myers, B. Getting more out of Programming-By-Demonstration. In Proceedings of CHI ´99. ACM, Pittsburgh PA, 1999, pp. 442-449
6. Mereu, S. & R. Kazman. Audio enhanced 3D interfaces for visually impaired users. Proceedings of CHI '96, pp. 72-78 ACM Press. (1996)
7. Montemayor, J. Physical programming: software you can touch. Proceedings of ACM SIGCHI, ACM Press, pp. 81-82, March 2001
8. Sánchez, J. Interactive 3D sound hyperstories for blind children. Proceedings of ACM CHI ´99, (1999). Pittsburg PA, pp. 318-325
9. Sánchez, J. AudioBattleShip: Blind learners' collaboration through sound. Proceedings of ACM CHI ´03, (2003). Fort Lauderdale Florida, pp. 798-799
10. Sjostrom, C. Using haptics in computer interfaces for blind people. Proceeding of the ACM CHI ´01, Vol 3, 1, (2001), pp. 245-246
11. Siegfried, R. A scripting language to help the blind to program visually. ACM SIGPLAN Notices, Vol 37 (2), pp.53-56, February 2002
12. Smith, A.C., Francioni, J.M. & Matzek, S.D. A Java Programming tool for students with visual disabilities. Proceedings of ACM ASSETS ´00, pp. 142-148, 2000