Irys: The Focus Oriented, Auditory Online User Interface

Andrew AbuMoussa
Independent Study
Department of Computer Science
The University of North Carolina at Chapel Hill
2009

Abstract:

As the push for the graphical representation of information continues within computer science, the most prevalent solution to providing the visually impaired with a usable computing system continues to rely heavily on screen reading solutions. Though this solution has provided some benefit, many concepts are lost in the cross sensory translation or mapping. For example, users of screen readers lose the concurrency afforded by having multiple graphical windows open simultaneously, and have a hard time discerning unprompted changes of focus. As interface designs continue leveraging graphics, the attempt to translate the dynamic interface's presentation into a serial auditory stream becomes polluted with inefficiencies and downfalls.

The goal of my work has been to explore methods which tackle the many aforementioned pitfalls of current graphic user interfaces. Leveraging this exploration, I have begun to address them by creating a cross platform auditory interface that is universally accessible through the Internet.

By rethinking the use case for modern computers, I have created a task based system, relying on concurrent speech processes to communicate the state of the system to the user; a vividly different and more useful approach from simply communicating the state of an OS's GUI.

Introduction:

The success of the graphical user interface can best be seen by simply observing any personal computer. Their [GUI's and personal computers] existences go hand in hand. Currently, the GUI is the mechanism that affords the personality of personal computers. Yet, as development in graphics continue to advance, the gap between accessibility and function continues to grow. The hardware driving the graphics continue to afford luxuries such as multiple monitors and graphic manipulations that improve productivity for sighted end user. As these effects are adopted in mainstream computing (e.g. multiple desktops in Linux, or their spaces equivalent for Mac OSX) their function and purpose get lost in translation resulting in a more confusing and less productive work environment for the blind.

In his dissertation thesis, Peter Parente notes that before the advent of the GUI, text based command lines afforded both those with visual disabilities and sighted users with the same level of usability. The command line interfaces textual basis allowed any method of screen reading to directly present any information from the screen to those with visual impairments. The push to GUIs stripped the blind of this simple solution as as the underlying architecture transitioned from a textual basis to a pixel mapping.

Though graphics are continually evolving and functionality continually being introduced through GUI's, accessibility solutions do exist. One attempt at a solution by T. V. Raman has been Emacspeak, a wrapper for Emacs, that enables the text editor to function as an audio enabled system. Functionality is then provided in the form of modules that can be installed on any Emacspeak installation, either by leveraging the emacs's inherent text interface or the command line interface accessible to the software. The utility of Emacspeak is derived from the applications natural text based interface, and thus lends itself to a true translation to spoken word. In that regard, Emacspeak vastly improves upon the screen reading solution by adding functionality to a text based interface, but in doing so, fails to allow the user to benefit from existing graphical interfaces entirely as it requires a custom installation for each user.

After observing the main trends in accessibility solutions, a fuller solution to the problem at hand came as the benefits, as well as the pitfalls, afforded by graphics were understood. In doing so, I enumerated a few key values that any good audio interface should incorporate:

A. Graphical interfaces allow for a complete static state; graphics can maintain state by acting as a photograph, providing a snapshot containing visual queues that a user can leverage to return to in an attempt at resuming a work-flow. A well designed audio-interface should have the capabilities to assist the user when returning to a given work-flow (e.g. scanning features, summaries, context information).

B. Graphics are restricted by the size of the screen, but the placement of application windows allow for an infinite stack to form (and even some OS's provide multiple desktops to increase x,y real-estate). An audio-interface should have no dependency on screen size or presentation. What it should retain from GUI is the ability to trace the work-flow of the user as they switch between tasks, and handle that trace in a way that is conducive to productivity.

C. Graphic interfaces afford tools such as icons and shortcuts that provide functionality. These prevent the user from having to memorize commands and allow the tool set to be available at any time. Audio interfaces should relieve the user from having to memorize functionality as well.

D. Concurrency between applications is afforded by allowing each task to have its own window. An audio interface should also provide a means to discern between different applications that is intuitive for the user.

E. One of the greatest pitfalls of GUIs and solutions to accessibility issues is locality. Each

requires a custom installation. In that regard, a key feature of a well planned solution is system independence and global accessibility.

Irys: Motivation and Design

The goal of this independent study has thus been to develop an audio enabled interface that focuses primarily on the benefit to blind people. In so doing, an interface can be designed that not only benefits those that can't see, but with adaption, can even come to help those sighted people as well (imagine having a web enabled console in your car's dashboard that did not require a monitor, yet afforded much of the functionality mainstream interfaces provide). Following is a discussion of design decisions and implementation considerations that have led to the creation of the proof of concept of Irys.

To provide the widest base of support, an online interface was chosen. A web-based interface guarantees simultaneous cross operating system support since the software currently runs on any system with the Mozilla Firefox browser and a plug-in that enables Javascript access to any OS's text-to-speech software. Another benefit of having an online based system is that updates get pushed globally so that version support issues can be disregarded since every user will run whatever is implemented by the server. Finally, the web-based nature of this interface relieves the user from having to custom install the functionality on every computer used.

Modularity & Ubiquity

As a linux user, there is a certain elegance observed in the command line prompt. By having a set of tools that carry out a single function, but provide functionality as a whole through nesting to produce complex commands, is something that Irys hopes to implement. The reasons being two fold: provide the user with a powerful and flexible interface, while freeing the programmer from having to worry about complex behavior since nesting of functions and outputs are assumed to provide this type

of functionality.

Let me pause the discussion of this command line tool set, since it then becomes a matter of deciding what functionality to give the user and then discuss the issue of navigation within this type of operating system. I chose to implement the highest level of navigation through progressive searches. By this, I mean, there will be one main key binding that works to bring up a search dialog from anywhere in the interface, and that the search will be carried out *as* the user provides any information. Rather than having the user construct a search string that they think might work, the interface will return any and all the possibilities that match the given search parameters as they are provided. In this way, file management can be hidden from the user, since the user will be able to search documents based on filenames and by the content within the document, and as matches are found, the user will be alerted to the number of matches, and once the result set has been reduced to an arbitrary number, can then proceed to enumerate them to the user. This provides a Google like functionality for the interface, while at the same time, improving on the model presented by Google, by having continual feedback to the user through its progressive search implementation.

The scopes of the search can then be defined by the state of the current focus (focuses are discussed later). For example, if a user were to call up a search dialog from an IM client, the search would know that any matches within the client would have a higher priority over any global results (ie files found, or matches within any data structure of another focus), and would be presented to the user accordingly. In order to provide this type of functionality, the interface would assume that the search was intended for that application, and hence display results with a higher priority, and then each focus could define another set of rules with regard to how the results are to be presented to the using a priority model. For example, it becomes an issue for the programmer to map the search data structure, to comprehensive information (i.e. if a search from within a calendar focus was carried out, it would not really help the user to know they have an entry on 2,2,32:14:30, but it might be better to let them know that “On Thursday February 2nd at 2:30” there is an entry that matches a given search parameter).

A design choice about the search has been to provide a unified presentation of results to the user in the form of a result field within the search dialog rather than taking an emacs or vi approach where a search takes the user to the first matching occurrence (after which the user can cycle through all the matches). This was an arbitrary decision, as having the interface cycle through focuses that bring up the matching occurrences in their respective focus would be a neat feature, the scalability and utility would have to be determined in the field, but not a subject of preliminary design.

I will return to my discussion on the utility of modular functionality later, but for now, I hope it suffices to acknowledge that it would be a nice feature for power users to have, while it wouldn't hurt the usability of the system for first time or novice users.

User Focuses

Functionality will be provided by a what has been coined as a focus. A focus is a small module of code that is used by the interface to carry out a single task (ie. A calendar focus). At first, it was thought that this online interface would adapt applications in a way that would allow them to be audio enabled, but after more thought, this approach proved to reincarnate the problem at hand, namely that, complex applications (depending on graphic user interfaces) do not inherently lend themselves well to be audio enabled. Instead, these focuses would allow the user to perform a number of given tasks. A focus, in essence, strips the complexity afforded by GUI applications, while providing the functionality that many similar GUI based applications afford, in a simple linear manner. By focusing on a the completion of a single task, the problem of enabling audio becomes a simple matter of deciding what the designer of the focus wishes to communicate to the user since the actions taken to complete the task are assumed to be linearly mapped by the programmer.

A few focuses that were explored were IM and email clients (tackling the problem of synchronous vs. asynchronous communications), a calendar, and a word processor. The type of word processors that beg to be implemented by this type of interface are text based editors such as VIM and

Emacs. The problem that arises is the steep learning curve required to get the full use out of the system, complicated by the lack of audio enabled documentation (not to mention the need for dynamic documentation as the user starts personalizing the commands) thus the implementation would be a small challenge that would afford great productivity to the power user, but it did not seem like it would provide the new user with as much utility as a much simpler word processor. Secondly, their implementation would be a deviation from the idea that the user should have only one single simple focus, and not an application in front of them. Ultimately, I believe it would be a great tool to provide, but not initially.

To return to the task based focus, it became clear that a word processor could be stripped down to the use of a text field, where the user could type and create a document or file. The formatting of this file would be done by the server, and as any word processor, be a task hidden from the user. To cater to the blind, the focus should be enabled to speak to the user any input as well as provide scanning capabilities, such as reading from the beginning of any line, or the first complete sentence (or two) from a given paragraph, or put differently, be conducive to navigation. In designing this single focus, global design considerations became apparent. The two main problems that need to be tackled by any focus design is that of user navigation (both during and resuming work) within the focus and it's data as well as the linearizing of the task it is trying to complete.

Implementation of Features from Graphic User Interfaces

Earlier in my discussion of the history of GUI design, I enumerated some features that enhance the functionality of the overall system while at the same time increasing a given users productivity. While brainstorming the design of Irys, I have come up with solutions that would both, provide similar functionality to both sighted and blind users and improve upon the design to cater to the problem of translating the interface into a serial auditory stream.

The first, and arguably the greatest, feature of graphical user interfaces seems to be the inherent

ability to maintain a photographic state by which a user can return to and resume work based on visual cues from scanning or reading. This has by far, been the most difficult feature to reproduce in the auditory realm since the transience of auditory stimulation is no where near that of the permanence afforded by static screens and the state maintained by a display. To those sighted users, the auditory interface provides this functionality by having a graphical component, but the users for which this system was developed for, need a different solution.

Attacking this problem, I've considered the differences in providing this type of state depending the last used focus (i.e. resuming a chat conversation, which is mostly a linear task, may require the user to have a number of previous lines repeated to remember where he/she left off, while resuming a word processing task, may require a much more intricate return pattern, one that involves scanning paragraphs, rereading certain ones, etc). Through time and development, maybe this type of state will prove to be a transient loss, as different patterns of use prove to be more natural, or it may prove to be a crucial component of all successful interfaces. Currently, this problem needs more time to form a more efficient global solution, but currently, work resumption and state maintenance is a responsibility of each focus and is a function of the task being completed.

Continuing with the features of GUI is the realization that a screen's size limits the number of windows that can be displayed (this can be ameliorated by an OS's integration of features such as spaces and multiple desktops) but what really allows an interface to be conducive to productivity is the infinite stack of application windows that any GUI can manage in the z-direction of the desktop. As applications are opened, the most recent moves to the front, and as you close windows, you traverse through this stack in a FIFO manner. Mimicking this functionality, Irys will keep a stack of focuses last accessed, allowing the user to return to the last focus that was opened, while keeping pointers to the previous focuses as new ones become opened. In that regard, each focus can be running on the client's machine, though not visible or accessible until attention is returned to any given focus. This allows multiple asynchronous background notifications, levering an OS's ability to use multiple concurrent

voices to alert the user of ongoing changes, while at the same time, implementing the single-use focus system that provides the fluidity of the text-to-speech model. The choice which prevents or allows the user from having multiple focuses accessible at any given time is a topic of discussion in Jef Raskin's book entitled “The Humane Interface”, where he cites psychological reasons that would argue against that luxury. This is a detail that could very easily be changed, so it is not given too much thought at the current time. What is important though, is that the flow of the clients use is stored in a data structure that allows the interface to return to the last accessed focus after closing any given focus. Raskin argues that this is an integral productivity feature for most humane interfaces, and simple logic suffices to make its case.

Another defining feature of GUIs are graphics, or the ability to display icons that carry out tasks or functions. This feature is perhaps the greatest cause for the success of the GUI since it frees one from having to memorize a set of given commands, but rather provides a mapping that can be used at any later point, to execute commands quickly and efficiently. Since the target user base doesn't have sight, one solution to icons, is to assume that they do nothing but clutter the user space and hence have no place in this type of interface. Though current interface design argues that any hidden commands available to the user can be assumed to be invisible if not non-existent, I believe that one level of indirection could solve this issue. It could be argued that every focus would use a unique key binding for every action, but then the user is forced to memorize an extremely large set of commands. Another level of indirection could be that whenever a user wants to execute a special command specific to a given focus, a key binding would bring down a menu specific to the focus. The user could then traverse this type of menu using arrows, though these menus should be relatively simple. There is still a lot of thought that needs to be afforded to this type of issue, again something that would get ironed out through implementations.

The presentation of concurrency really concludes the great benefits of GUIs. This also is where the greatest pitfall for screen readers lies in current systems and implementations. For example,

unwanted pop-ups are not treated as special cases of user interactions, hence users of current screen readers can't account for the sudden change brought about by pop-ups. With this interface, pop-ups are prevented by not allowing any thing to pop up in the first place, but rather, any focus change results in one section of code becoming visible or rather, gaining focus, and the other becoming invisible. The uni-focus implementation of Irys prevents the system from confusing the user, but concurrency is maintained as the underlying logic of any given focus is still executing on the clients' machine. In that way, any focus without the users attention, can request to alert the user of a given change through concurrent speech while the user's sanity is preserved.

Focus Deployment

A lot of the theory has revolved around the interfaces integral use of focuses to provide functionality. As a web based system, the question of how these focuses are loaded by the user, and used during a session, begs to be answered. The first iteration of the interface will leverage the fact that a user can subscribe to any number of focuses and that these focuses are simply javascript files that have a unified visual and auditory interaction with the client's computer and that they interact with the site in a defined manner. The server then acts as a store for these focuses, and during a login, the server sends the appropriate focuses to the client's machine and the client can then manage their interactions. In that regard, an API for focuses must be developed as well as a method for accessing the available focuses and subscribing to them.

Miscellaneous Functionality

The current interface design has laid a backbone for the interface. What remains is to actually implement the design patterns and add functionality, and see what works and what does not. On top of the basic interface design, some convenient functionality could be a universal “Back” & “Forward” Stacks. These would function as a global undo or redo button for the user, that also have the ability to

change focuses hence doubling as a navigation tool. One such use could be a user returning to the site, logging in, and then pressing back to see what they were doing last before ending their prior session. In Raskin's discussion, the removal of a universal redo and undo buttons from the keyboard was questioned, since the two buttons could relieve a huge time sink caused by the desire to reproduce a prior action that was accidently omitted.

The “What just happened?” button would also be an interesting approach to a user's manual. This key binding would require every command executed by the interface to carry an explanation of what the command did, and what caused this command to be executed. It would put a larger burden on the programmer having to document every action thoroughly, but in that way, a user could be traversing the undo/redo stack, learning why certain actions were carried out. It's an interesting feature to consider as it would be a nice learning utility, but it's inclusion is yet to be determined.

Analysis: Utility

One field of interface design that I found to be interesting was the analysis of given systems and interfaces. This branch of analysis has come up with given standards for GUIs, which could be a standard of comparison for the work that is to come. By far the greatest delay in any GUI is the time it takes the user to visually navigate the interface and locate or move to a given point of interest. As I design the online interface, this would be an integral tool in measuring its success, and could be used as a standard for which to aim. The ultimate goal for this project would be to create an interface where the times of uses become comparable to those of a sighted person using a GUI rather than the overly simple goal of creating an interface that is slightly better than given screen reading solutions.

Future Work

Currently, Irys has an audio-enabled login and registration page demonstrating the ability to create web applications that are audio enabled. What remains is then to create the focuses for the end

user to interact with. Ultimately, the visual presentation will be aided by the use of Dojo. Some features I would like to add after implementing the interface would be customizations for power users to map out keys (which would dynamically change the user's manual for that user). Ultimately, the goal for Irys is that it would be completely audio enabled in the sense that the interface could be given audio commands and that it could be a system that works on any device, from phones to complete PCs.