REDUCING REPETITIVE DEVELOPMENT TASKS IN AUDITORY MENU DISPLAYS WITH THE AUDITORY MENU LIBRARY

Parameswaran Raman, Benjamin K. Davison, Myounghoon Jeon, and Bruce N. Walker

Sonification Lab
Georgia Institute of Technology
430 Cherry St.
Atlanta, GA 30332
params.raman@gatech.edu

ABSTRACT

This paper explores the process of auditory menus research. Several parts are tedious tasks which must be repeated for minor changes to the experiment. Fortunately many of these parts can be automated with software. We present the Auditory Menu Library (AML), a tool for simplifying experiment construction. The AML provides a cross-platform, configuration-based turnkey solution to studies involving auditory menus.

1. INTRODUCTION

This paper explores auditory menus research from a process standpoint and describes an extensible multi-purpose library to help perform similar auditory menus experiments in the future. By reducing the costs involved with experimental development, we will expand the possibilities of research in auditory menus.

The first part of our investigation explored the various research elements of interest in auditory menus, such as submenu behavior or sound types used. This follows the footsteps of Yalla and Walker [1], but we focus on what has been explored experimentally instead of what conceptually is important in menus.

We then studied the general parameters that researchers might be interested in configuring across various sets of experiments and the steps that are required to prepare for user studies. These will give us an understanding of the breadth of study-based functionality that a supporting software tool should possess. In addition, the repetitive nature of menu design, experimental definition, and platform-specific user interface descriptions begs for a more generic, automated process that will support study replication and extension.

This report summarizes our findings and describes a new tool: the *Auditory Menu Library* (AML). This research tool has been developed specifically to aid sonification experiments in auditory menus. The final section describes the system architecture of the Auditory Menu Library, explaining its design and implementation details, with relevant examples.

2. AUDITORY MENUS RESEARCH

Menu displays in a computer provide a list of choices. When selected, the system performs a particular task. Getting to the correct selection involves an interaction between the display and the user. Finding a particular item in a menu involves a consideration stage, which can take place in the user's mind but often is also a part of the display itself. In visual menus, this is typically done with a highlight over whatever menu item the user has their mouse hovering. Therefore, a menu choice involves both a pre-selection, or hovering, and a selection, some binary action that indicates that the user has decidedly picked the hovered item. In auditory menu display *pre-selection* of menu items is the most informative step. The actual selection often does not render the full representation of the pre-selection.

Auditory menus research has featured several components: speech; non-speech: auditory icons, earcons, spearcons, and spindex; auditory scrollbar; combinations of speech and non-speech sounds; available and unavailable menu items; visuals on or off; push and pull menus; different interaction styles; hierarchical menu structures; and broad versus deep menus. This section explores each of these components in more detail. For an analysis of visual menu structures and their applications to auditory menus, see [1].

2.1. Speech

The naive auditory menu is completely speech-based, activated when the user pre-selects and selects a particular item. Specifically, the item label (the visual text) and sometimes the item's properties (such as "unavailable") are "spoken" by the system. Commercial systems that speak menus use a text-to-speech (TTS) engine. For research, high-quality TTS or human speech is prerecorded for the menu.

Since visual menus are spatial, and since people can easily move their eyes to "scan" a menu, responses to searching for an item on a visual menu are relatively faster than the a spoken menu. Non-speech sounds alleviate this serial characteristic of speech-only approaches. However, Non-speech sounds are not typically designed to fully replace speech in a non-visual auditory menu, and often the study stimuli use speech in combination with non-speech sounds.

2.2. Non-speech sounds

The first studies in using non-speech sounds in auditory menus were attempted in the form of earcons by Brewster in particular [2, 3, 4, 5]. Walker et al proposed a speech-based sound called Spearcons that could be used to improve navigational performance in auditory menus [6]. This was followed by the conception of another concept called Spindex [7] which provides auditory equivalents to the visual concept of bookmarks used in a telephone direc-

Experiment	Device	Prog. Env.
Spearcon and TTS [12]	Desktop	Director, mobile
Spearcon and TTS [6]	Cellphone	Java
Spindex and TTS [7]	Desktop	Director, mobile
Spearcon, Spindex, and TTS, dual task [14]	Car head unit	C# Centrafuse
Auditory Scroll Bar [8]	Desktop	Flash

Table 1: Research with one-dimensional menus.

Experiment	Device	Prog. Env.
Spearcon, earcon, auditory icon, and TTS [15]	Desktop	E-prime
Spearcon and TTS [20]	Desktop	Director
Earcon and spearcon [19]	Desktop	Director
Auditory icon, earcon, spearcon, and speech [18]	Desktop	Flash & Java
Speech menu item availability [21]	Desktop Mac	Java

Table 2: Research with hierarchical menus

tory. Auditory Scrollbars were designed and evaluated by Walker and Yalla in [8]. With such a growing scope for research as evident from the previous citations, it also becomes essential to conduct exhaustive experimental analysis, evaluate and gathering feedback from the user community. The rest of the section focuses on some key elements of research in the wide domain of auditory menus, with a view to gather data that would be useful to help identify the prime objectives and development goals for the AML. Yet another work in non-speech sounds is auditory icon. Gaver's auditory icons provided a way to represent WIMP[1] elements in a non-arbitrary way [9, 10]. Natural sounds could represent incoming mail, folders, or a disk drive. However, not everything in the computer can be metaphorically converted to a natural sound; what is the natural sound of "Save as HTML"? The other non-speech sounds sacrifice the natural sounds for categorical and descriptive information [11, 12].

Earcons originated as a way to provide organization information as variables in the sound itself [13]. Elements of the item's role and position in the menu structure were represented. This would arguably facilitate the user's navigation of the menu space, particularly if visuals are not available [11].

The definition of "earcon" seems to encompass any brief, organized sound that is not natural or speech-based. This simple worldview could categorize all menu sounds as having components of earcons, auditory icons, or speech. However, spearcon and spindex are hybrid concepts which provide structural information like an earcon but are rooted in a verbal interpretation of the element.

Spearcons gain some benefit of speech information while reducing the time cost of listening to speech [15]. Speech is an effective way to convey information. Spearcons attempt to maintain elements of the phonemes that would be present in the standard oral output. Spearcons are compressed speech using a type of selective sampling of the speech based on the SOLA (Synchronized Overlap Add Method) algorithm [16, 17], which produces the best-quality speech for a computationally efficient time domain technique.

Spearcons might be useful in more familiar menus, such as a personal cell phone address book, but take some time to learn. Therefore, they are typically placed along with spoken text following the spearcon. The user can skip the full speech once they are familiar with the items. The utility of spearcons has been evaluated many times [12, 14, 18, 19, 6].

Spindex is an indicator of the user's position in a sorted alphabetical menu. Typically this is manifested by the first letter of the selected word. Accordingly, for a name "John Smith", the spindex cue is /dʒeɪ/ or simply /dʒ/ (or /es/ or /s/, depending on sorting). These cues provide the user, an indication of his current position in the list and help him navigate to the desired item much more rapidly. Thus the spindex is most useful when going through large sorted lists [7], since the user could make an educated decision as to move up or down based on the current spindex cue.

[1] Windows, Icons, Menus, Pointers interface.

2.3. Auditory Scrollbar

An auditory scrollbar is an analog to a visual scrollbar: as the value in the 1-dimensional range changes, the scrollbar changes in pitch. The auditory scrollbar can be designed in four different ways - using single tone, double tone, proportional grouping and alphabetical grouping [8]. The pitch polarity is adjusted based on these approaches. Auditory scrollbars may be useful in menus like a Font menu that has hundreds of fonts listed in alphabetical order where the user cannot navigate one by one. In such cases, it might also make sense to combine auditory scrollbars with other non-speech sound variants like spindex to enhance the user search. Auditory Scrollbars are therefore, flexible enough to be used in conjunction with other non-speech sounds in order to add value to the user experience.

2.4. Combinations of Speech and Non-speech sounds

The sounds mentioned in the previous sections are often combined in research studies. Spearcons, for example, are typically combined with text-to-speech (TTS), so that a novice can still use the TTS, while someone familiar with the list can leverage the spearcon for quicker selection (for example, in [12, 14]). This brings up several considerations:

- Is there any temporal overlap of items? Typically they are presented serially.
- What sort of interval (quiet gap) should be placed between the sounds?
- In what order will the sounds play?
- Should the display change with experience?
- How should menu item properties be represented?

Quite often, studies are comparing one approach to another, so ensuring an apples-to-apples ordering and interval is a critical aspect of the study.

2.5. Intervals and Ordering

An auditory menu item is always likely to be supported by a diverse set of non-speech sounds like spearcon, spindex, TTS and auditory scrollbar. Since there is a possibility that researchers might want to test all of them together, it is very important that they conform to some standard order in which they are played. For

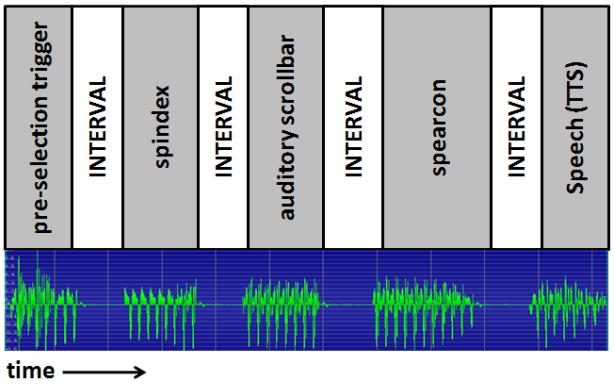

Figure 1: Order of non-speech sounds.

instance, using a spearcon, spindex, TTS and auditory scrollbar on the pre-selection of a menu item might have an order as shown in figure 1.

There can be notable intervals between each of these sounds which is a value that is configurable. Varying this interval value and sometimes the order might lead to interesting results in auditory menus.

2.6. Unavailable menu items

Auditory menus tend to add information such as *unavailable* or *dimmed* to the menu items which are disabled. An immediate example for this is the screen reader software that ships with the Macintosh operating system. However, there are also other rendering possibilities such as using a lower voice or a whisper voice. One can also switch between male and female voices to evaluate their respective effects on menu usage and performance [21].

2.7. Visuals On/Off

Visuals On/Off is an essential experimental setting for almost all the auditory menu experiments which allows the users to get a real-time context of how auditory cues help them where no visual cues are present. Often, when the visuals are on, we tend to rely on them and not focus on the auditory elements fully. Having a way to turn visuals off in almost every auditory menu experiment is thus crucial for better results.

2.8. Push/Pull Menus

Another useful way to classify the auditory menus could be push menus and pull menus, which are studied in context-aware systems (e.g. [22, 23]). These can plausibly be applied to auditory interfaces as different ways of deriving information from the menu items. In the case of push menus, the menu keeps reading out the menu items (using TTS along with the non-speech sounds and other auditory cues) in a specified order; it loops through all the menu items. The user can then select the desired menu item when it is being spoken. This is different from the more common pull menus, where the menus items are played out to the user based on navigation navigation (eg: cursor button presses). Or, in other terms, the user decides which menu items to pull out. In our experiments, we make use of these two variants very frequently and hence, need a way to represent this configurability.

2.9. Interaction Styles

Auditory Menus have been tested on the desktop [12], on mobile phones [6], and with in-vehicle navigation systems [14]. These experiments involve users searching through a one-dimensional menu list (such as a phonebook or MP3 song list) using up and down arrow keys for a particular target name. However, since many smart-phones today use advanced interaction styles such as flick, finger-gestures, tap and wheeling on a touch screen device, the research community should consider how these interaction styles affect the auditory display of menus. There should also be an easy way to customize them as per needs and switch from one style to another for demonstration purposes.

2.10. Hierarchical Menu

Hierarchical menus are more complex to handle because of the variety of information that they contain. Some properties of hierarchical menus include:

- Number of items in a menu
- Available/Unavailable state of the menu item
- Accelerators/Hot Keys for the menu item
- Grouping of menu items and Separators used
- Type of menu item (does it invoke a dialog, is a sub-menu or top level-menu?)

The key challenge therefore is to incorporate all of these characteristics successfully into auditory menus. More importantly, being able to mix and match these features and test them across different platforms and devices is essential for edging closer to practical auditory menus.

2.11. Broad versus Deep Menus

The use of particular displays in a menu may depend on the menu structure. One basic division in menu structures is broad versus deep. There have been many explorations into broad versus deep visual menus [24, 25, 26], and some in auditory menus [27]. Jeon and Walker suggest that very broad menus have not been typically considered [7], and their spindex solution is similar to a visual analog of a visual letter of the current item appearing on the screen, as seen on some iPods when scrolling through long music lists. Changing the display of different structures of menus, visual or auditory, is an open area of study.

3. AUDITORY MENU RESEARCH CHALLENGES

Based on the review above, there are several elements that appear important to a research-oriented developer of auditory menus.

- There are many auditory menu sound types. Selection and generation of the appropriate comparisons of sound types is a key component to testing them. This includes combinations of sound types including intervals.
- There are many menu properties, such as availability, accelerator, and submenus. While most of the research considers only the structural and functional role of a menu item, many other properties are commonly used in real applications and need to be considered.

- Menu structures can be very broad(i.e., 1 level of 1000 songs), or fairly deep(e.g., 5 levels of settings with 2 to 8 items in each category). The structural definition of the menus is a key component to describing the results of research, since the structure determines, in part the ideal human performance of the task. In addition, certain auditory menu sound types are designed with a particular type of menu in mind. Intuitively a hierarchical earcon is designed for a deep hierarchy, while a spindex succeeds at long, single-dimension lists.

3.1. Roles

Auditory menus experiments require the following roles:

- an experiment *architect* who designs the study. The requirements are made by this role.
- a system *developer* who produces the software framework in which the study is run.
- an *experimenter* who actually runs the experiment.
- a data *analyst* who determines what the study shows.

Multiple roles may be held by a single person, such as architect and analyst. It appears that the system developer is often a different person from the experiment architect, who probably has a more deeper insight into human factors and usability.Therefore, communication of experiment goals is a critical component in attaining what is needed.

System development is necessary but arguably it doesn't need to be as complicated as it typically is. For example, if an experimenter needs two studies, one with spearcons and one with earcons, the menu structure and logging system along with the rest of the system can basically be the same.

3.2. Non-developers

The objective of building the library for auditory menu experiments is to essentially bypass a need for a programmer. Most of the menu definition is configurable. The average time required by a person (not well-versed with programming) to create a menu activity capture program, configure log files, traces and reports to capture the results, and then run the trials should be as minimal as possible. For instance, if the experimenter needs to add an auditory scrollbar sound to her auditory menus, it should be as easy as adding a line in a configuration file.

Currently, most of the tools developed are tightly bound to the developers. For example, an application showcasing auditory menu concepts on a Nokia phone with a phonebook list of items cannot be easily tweaked by changing the names and adding new sounds to it by a non-programmer, because this requires source code modification, compilation, packaging, porting and so on. At best, the names can be configured in a separate file, but the experiment cannot be then run on the desktop or a different cell phone platform. In addition, development in the same platform and programming language may be done in parallel because of lack of understanding of another's code base. This division leads to highly repetitive system designs. Most of the systems described in Tables 1 and 2 were made independently of each other, not sharing a common basic structure. As a result, there was no feedback cycle and similar amount of effort in terms of design and programming was spent in both of them. This could have been avoided by sticking to a generic framework that is reusable over and over again.

Be it Macromedia Director, Java or in-vehicle applications, a change in the program environment to conduct a user study seems a tough job for a non-programmer. Also, it doesn't make sense to keep learning new programming languages just for the sake of modifying the application for re-use. It is unnecessary overhead and time consumption.

3.3. Programmer Effort and Time

"Why do something in two days that will take two months to automate?" Ever since the inception of the ideas around non-speech sounds, several applications were developed to test the usability and evaluate the concepts developed. However, each time a new application was developed from scratch; this resulted in duplication of programmer effort, often with similar pieces of code being rewritten. There is code redundancy and unnecessary delays every time an experiment is performed. Instead of evaluating entirely new systems, developers can be put to use by adding concepts and platform support to a larger auditory menus tool.

3.4. Multiple Devices and Platforms

Beyond accessibility for the visually impaired, auditory menus may provide extra help on platforms with lower visual space and attention than a traditional desktop interaction. Auditory menus have been evaluated on the desktop, in mobile devices, and with in-vehicle systems. Thus, simulations need to be designed on each of these several types of platforms that demonstrate the features available and help the users evaluate them. This is again a repetitive task because though the research questions might be similar, the particular software implementations vary from desktop to a mobile device and to a vehicle (particularly in the user interface). In addition, a replication will work best if a similar process and identical data and stimuli are used. It would be impressive if there were a way to reduce this implementation time and if researchers could have a base toolkit that would expose much of the generic features, which have to be merely extended with minimal effort.

3.5. Replication of Experiments

A research tool should support the replication of a past study. By separating the data from the user interface, different researchers over different places and times can run the same experiments if they have the same data and environment set up. Building auditory menus involves putting in pieces of code, wav files, adding configurations, etc. There needs to be a way to efficiently represent the auditory menus in its entirety without any loss of information and experimental data, and migrate this to any other device. For example, once you create a hierarchical auditory menu on a Macintosh notebook, you should be able to save it in some text file format on the hard drive and use it to load the menu back on a mobile device. It is reasonable to aim for a stable state where non-programmers could store auditory menus simply in the form of plain configuration files and play with it to generate different types of menus, add hierarchies to them, modify the sounds from one type to another, and initialize their experimental settings.

3.6. Existing Toolkits/Libraries for Auditory Menus

Very little prior work has been in this direction to build a toolkit application to aid research auditory menus in particular. Mynatt

and Edwards created a process and a software tool called MERCATOR designed to map graphical user interfaces to equivalent auditory interfaces [28]. This work essentially describes how an accessibility tool like JAWS creates accessible spaces. However, Mynatt and Edwards's was a much broader study not covering the specifics of auditory menus. Likewise, Brewster [29] and Davison and Walker [30] provide audio toolkits that extend user interface libraries with the concept of sound. Again, their focus was more general than specific auditory menu design. In addition, these tools are intended for use by end applications, and not specifically a research study, so considerations such as latency, logging, and quality of stimuli are different. There is also need for a data structure to represent an auditory menu of typical hierarchies and make it used in experiments. Thus, the key jobs of Auditory Menu Library are to automate the repetitive task of system construction and to model menu structure. It helps reduce the time between wanting a system and producing it.The next section discusses the system in more detail.

4. AUDITORY MENU LIBRARY

The Auditory Menu Library (AML) is a generic library currently in Java that helps its users represent the concepts of auditory menus using a data-structure, replicate them across diverse platforms and use them for experiments in a much more efficient manner. The AML has been designed considering the varying programming capabilities of major stakeholders involved in auditory menus research. Experiment Architects could just interact with its data and use it for customization to suit their experiments. On the other hand, a System Developer could aim to leverage its Application Programming Interface (API) to define more complex structures. The AML is defined as:

- A YML (a human-readable configuration file) document definition that describes the structure of the menus and settings to be used within the program. This defines the scope of acceptable YML files: those that can be turned into auditory menus by the AML.
- Menu objects that define the menu structure.
- A YML parser that converts a YML file menu definition into the menu objects.
- A hook for user interfaces to turn menu concepts into visual and auditory menus.
- A starter library that has converted the menu concepts into menus on a device like desktop computers and mobile phones.

4.1. Development Goals

There were several goals in creating this software. First, it would be transparent to a novice programmer. Other than working with configuration files, there is little he would have to do to bring up an auditory menu of his choice. Creating another modified menu would be as trivial as copying fragments of an existing menu representation and modifying its item names and other parameters.

Second, for the programmer, it offers methods that he could use to create menus, menu items, menu hierarchies and pack them appropriately into any structure he desires. He could also choose to add auditory information to it, or even extend it to incorporate a new research concept. In addition, and third, the library is extensible in that new menu components can be incorporated into it

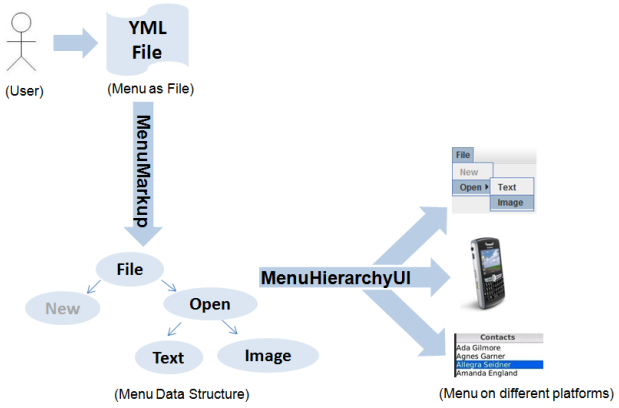

Figure 2: Process of creating menus using AML.

by relating them to the generic menu types. It is designed as a set of abstract Java classes which can be extended to suit the evolving requirements. This is clearly a one-time programmer effort, after which the architect, experimenter, or analyst could leverage its benefits. For example, defining a nearcon involves the following steps: Create a nearcon concept (Java class), explain programmatically how a nearcon works, and add nearcon parsing to the YML parser. What *doesn't* need to be done is a redefinition of the entire user interface, menus, or other audio interactions.

Fourth, the library is robust and portable. The architecture has been made so modular that the menu markup and menu model are totally loosely coupled. This helps since, a programmer would want to literally markup any menu that he has modeled and vice-versa without any glitches. Java was selected as the language since it has a strong cross-platform appeal on desktops. Current work is expanding support to different mobile devices. If a device isn't supported, the developer only needs to create the user interface explanation; the menu concept and YML files remain supported by the current AML.

Fifth, the library has the ability to represent most of the experimental scenarios in auditory menus with an emphasis to reduce developer intervention. As the science of auditory menus progresses and grows with more advanced features, this library can adapt itself and reflect all of those concepts as well.

To summarize, our focus is on the nuts and bolts of building the experiment, including:

- The roles of study development.
- Tools available for each role.
- The software structure that needs to be in place to show the menus and to log data.
- The stimuli creation.
- Log file structure: sufficient information, portable to other programs for analysis.

4.2. Architecture/System Design

The AML has been designed with the objective to achieve the above mentioned development goals and facilitate easy modifications to it in the future. What this essentially means is that all the components of the AML should be as loosely coupled to each other as possible. This would ensure that enhancements made in one do

not negatively affect the others. This section discusses many of the key components.

4.3. Menu Model

The AML deals with a variety of platforms and kinds of menus. It is therefore natural for the menu components to be diverse and different. The menumodel is a component that provides the ability to design new hierarchy of menus and represent them as auditory menus.

- **MenuType**
  MenuType is a base class (abstract Java class) from which the basic properties could be inherited to build customized and newer menu components. In other words, the MenuType defines the basic structure and the sub components under it define the functionality. This kind of design also facilitates better abstraction, as we could push the more generic features to MenuType, thereby maintaining the overall abstraction provided by menumodel. The underlying goal therefore is to extract the common features of all menu components and abstract them into this class. This prevents similar behavior from being treated multiple times amongst menu components. For instance, a menu item could be present either as *available* or *unavailable*. The same applies to a menu as well. Therefore, this common feature could be pushed up to MenuType that collects all commonalities.
- **Menu**
  Menu is a component derived from MenuType which represents an auditory menu. It is a collection of items of MenuType. This can be used for top-level menus, sub-menus, contacts list as seen in a mobile device and so on.
- **Menu Item**
  MenuItem is a component derived from MenuType which represents an auditory menu item. It defines most of the auditory menu properties and represents the user actions on selecting them.
- **MenuHierarchy**
  MenuHierarchy contains the menu structure defined in the menu model and is composed of a tree of menus and menuitems. It can be visualized as a single package that bundles all your auditory menu information and it can be then viewed on varying devices/platforms. Once a MenuHierarchy is built using the AML API, it becomes more convenient thereafter to edit it by just modifying the YML file representation of it.

4.4. MenuHierarchyUI

MenuHierarchyUI defines a way to render the MenuHierarchy into a UI specific to the device/platform. This exposes a method named *buildUI()* to accomplish the translation. For example, in the case of a desktop application, we could derive a specific implementation of MenuHierarchyUI termed as *SwingHierarchyUI* which uses Java Swing libraries to render the MenuHierarchy on the screen. Likewise, *BlackBerryMenuHierarchyUI* can be used to put the MenuHierarchy on a Blackberry phone screen. This is important because situations would be different on a mobile platform and we would need to use the J2ME specific UI libraries. Our aim is therefore to insulate the device specific technology from the core concept of auditory menus (which remains the same everywhere).

The SwingHierarchyUI is intended for desktop user interfaces, specifically desktops that use the Microsoft Windows, Mac OS, or Linux varieties of operating systems.

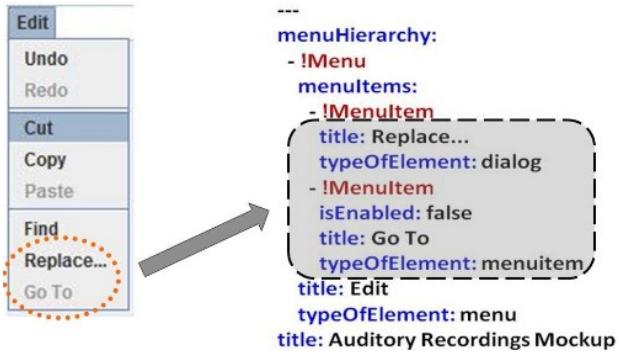

Figure 3: YML stores other meta-information about the menuitem such as typeOfElement, enabled state,etc .

4.5. Menu Markup

Menu Markup converts the YML files into their equivalent MenuHierarchy objects so that they can be displayed on different screens and devices.

- **MenuParser**
  MenuParser is a generic class used to read the YML representation of a menu structure and create a MenuHierarchy out of it. This MenuHierarchy could then be rendered based upon user's choice. MenuParser offers functions like parseFileIntoObject() to accomplish this. This could be inherited by several other classes like StandardMenuParser which could be used to parse an XML file into MenuHierarchy, or XMLMenuParser to parse an XML file.

4.6. YML as a Markup

YML has been adopted as the choice for markup over other commonly used languages like XML, primarily because YML is more human-readable with less meta-information and other unwanted data (refer Figure 3). This contrasts with an XML representation, in which most of the file space is consumed by opening and closing tags. YML's familiar indented outline and lean appearance makes it especially suited for tasks where humans are likely to view or edit data structures, such as configuration files, dumping during debugging, and document headers. It is extensively used in languages like Ruby and Python for storing user configuration.

4.7. Use Example

For example, the following code describes the process of constructing a MenuHierarchy named *Mockup*. As the very first step MenuHierarchy object *hierarchy* is constructed, by passing in a name and the visible state of the menu. Second, a *File* menu is created by adding two menu items *New* and *Open* to it. Third, the top level menu is added to the MenuHierarchy object *hierarchy*. AML provides numerous overloaded constructors for the programmer to set selective properties for the menu and menu items like enabled state, visible state, accelerator, etc.

The newly constructed MenuHierarchy object hierarchy, can either be saved as a YML file using the *dump()* function exposed by Yaml, or displayed on a user interface by using a custom MenuHierarchyUI object so that it can be heard or visualized. SwingHierarchyUI for instance, is a Java class that implements the *buildUI()* method of the MenuHierarchyUI abstract class, in a way so as to render the hierarchy object using Java Swing libraries on a desktop.

```
MenuHierarchy hierarchy = new MenuHierarchy("MH", true);
Vector menus = new Vector<MenuType>();
menus.add(new MenuItem("New", true, "menuitem"));
menus.add(new MenuItem("Open", false, "menuitem"));
hierarchy.addMenu(new Menu("File", menus, "menu"));
Yaml.dump(hierarchy, new File("FirefoxMenu.yml"), true);
SwingHierarchyUI swingUI = new SwingHierarchyUI();
swingUI.buidUI(hierarchy, true);
```

4.8. Salient Features

The library is extensible: a programmer can develop new menu components by extending the generic MenuType provided and adding specifics to it. This applies to other parts of the library as well like the MenuParser which can be extended based on the type of markup language followed.

Being developed in Java, AML is portable across most desktop platforms (Windows, Linux, Macintosh) and also can be used in Java-based mobile platforms such as Google Android and RIM Blackberry.

The greatest strength of AML is its support for configuration of its features. Choosing to turn visuals on or off, specifying which menu items are enabled or disabled can all be done by modifying the YML representation of the auditory menu.

Programmer intervention is required only if a new platform is encountered and AML needs to be extended to support or add features specific to it. Even that is a one-time task.

AML can be also be viewed as an effective research tool that helps you organize your experiments quickly and neatly. It provides support for:

- **Logging.** While running any experiment, there is an enormous amount of information a researcher might want to log about the user and his interaction with the tool. This could be response time, keystrokes, pattern of navigation, menu operations performed, etc. AML lets the person running the experiment choose the data to log and generates periodic log files which can be later examined for more details.
- **Debug Dumps.** These are files containing description about the auditory menu, its sub-menus, inner details, their state before the application crashed and so on. This turns out to be useful while troubleshooting.
- **Reports.** Data reports are essential to be generated for some experiments like auditory menu experiments on handheld devices to study how they perform with different user populations. Specifically, understanding how a new user adapts to auditory menus, measuring the change in his response time and his learning rate could help foster research in a greater magnitude. AML aids in capturing such information by tracking the user interaction.

4.9. Issues Faced

Several issues were encountered in the design and development of AML. It involved tremendous effort to make things generic bearing the various platforms in mind and devices that auditory menus might have to be used on. Even on using Java, numerous JDK version compatability and class file version issues had to be resolved to make AML run across desktop and mobile environments, because not all Java API are the same on all mobile platforms. Porting AML to the Android platform was a relatively easier task than Blackberry since Android is closer to the traditional JDK development supporting latest Java constructs such as iterators and generics. Infact, the way a particular task (like reading a file from the filesystem) is accomplished differs greatly from one platform to another. For instance, during AML's development phase, it was discovered that reading a file on Android seemed almost the same as on Desktop PCs but on Blackberry it was drastically different. Thus, a lot of functionality had to be implemented in different ways on each platform (leveraging features specific to the platform). As a result of these, the AML design had to be refactored by adding more generic features and interfaces.

Several delay issues were found while migrating AML to mobile devices because of the inherent limitations of the device memory and processor speed. As a result, the feedback of the application was sometimes quite different and inferior from the one as observed on a desktop. New solutions are being implemented to address issues such as these as they arise.

4.10. Demonstration Application

A sample application was developed that made use of the Auditory Menu Library to create auditory menu structures in the form of YML files, render it on the desktop user interface and also test the effects of applying the various sound elements like spearcon, spindex and auditory scrollbar to the menus.

This demo application built into the AML displays two different kind of menus - a hierarchical menu and a linear menu, both of them built from a single YML file. The hierarchical menu consists of active and disabled menu items, sub-menus and menu items that invoked a dialog. The linear menu consists of a list of phonebook contacts to which an auditory scrollbar was attached and its combination with other sounds like Spearcons and Spindex could be observed. It is also possible to turn off features selectively to emphasize other specific ones and derive conclusions.

5. CONCLUSION

This paper provided an overview of the history of auditory menus research. It explored the repetitive challenges faced, particularly during system development. Finally, we discussed the auditory menu library, an extensible Java tool designed to support experimenter and programmer development of research-oriented auditory menus. This software can be accessed at http://sonify.psych.gatech.edu

6. REFERENCES

[1] P. Yalla and B. Walker, "Advanced auditory menus," Georgia Institute of Technology GVU Center, GVU Technical Report GIT-GVU-07-12, Oct. 2007.

[2] G. Leplatre and S. A. Brewster, "Designing non-speech sounds to support navigation in mobile phone menus." in *Proceedings of the 6th International Conference on Auditory Display*, Atlanta, GA, 2000, pp. 190–199.

[3] S. A. Brewster, "Using non-speech sounds to provide navigation cues," *ACM Transactions on Computer-Human Interaction*, vol. 5, no. 3, pp. 224–259, 1998.

[4] S. A. Brewster, V. Raty, and A. Kortekangas, "Earcons as a method of providing navigational cues in a menu hierarchy," in *Human Computer Interaction*, 1996, pp. 167–183.

[5] S. A. Brewster, P. C. Wright, and A. D. N. Edwards, "A detailed investigation into the effectiveness of earcons," in *Proceedings of the 1st International Conference on Auditory Display*, Santa Fe, NM, USA, 1992, pp. 471–478.

[6] B. N. Walker and A. Kogan, "Spearcons enhance performance and preference for auditory menus on a mobile phone," in *Universal Access in HCI*, ser. Lecture Notes in Computer Science. Berlin: Springer-Verlag, 2009, no. 5615, pp. 445–454.

[7] M. Jeon and B. N. Walker, ""Spindex": accelerated initial speech sounds improve navigation performance in auditory menus," in *Proceedings of the Annual Meeting of the Human Factors and Ergonomics Society*, San Antonio, TX, Oct. 2009.

[8] P. Yalla and B. N. Walker, "Advanced auditory menus: Design and evaluation of auditory scroll bars," in *Proceedings of the Tenth International ACM SIGACCESS Conference on Computers and Accessibility*. Halfax, Canada: ACM Press, Oct. 2008, pp. 105–112.

[9] W. W. Gaver, "Auditory icons: Using sound in computer interfaces," *Human-Computer Interaction*, vol. 2, pp. 167–177, 1986.

[10] ——, "The SonicFinder: an interface that uses auditory icons," *Human-Computer Interaction*, vol. 4, pp. 67–94, 1989.

[11] S. A. Brewster, P. C. Wright, and A. D. N. Edwards, "An evaluation of earcons for use in auditory human-computer interfaces," in *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*. Amsterdam, The Netherlands: ACM, 1993.

[12] D. Palladino and B. Walker, "Efficiency of Spearcon-Enhanced navigation of one dimensional electronic menus," in *Proceedings of the 14th International Conference on Auditory Display*, Paris, France, 2008.

[13] M. M. Blattner, D. A. Sumikawa, and R. M. Greenberg, "Earcons and icons: Their structure and common design principles," *Human-Computer Interaction*, vol. 4, pp. 11–44, 1989.

[14] M. Jeon, B. K. Davison, J. Wilson, M. Nees, and B. N. Walker, "Enhanced auditory menu cues improve dual task performance and preference with in-vehicle technologies," in *Proceedings of the First International Conference on Automotive User Interface and Interactive Vehicular Applications*. Essen, Germany: ACM, Sept. 2009.

[15] B. N. Walker, A. Nance, and J. Lindsay, "Spearcons: Speech-based earcons improve navigation performance in auditory menus," in *Proceedings of the 12th International Conference on Auditory Display*, London, England, 2006, pp. 63–68.

[16] D. J. J. Hejna, "Real-time time-scale modification of speech via the synchronized overlap-add algorithm," Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1990.

[17] S. Roucos and A. M. Wilgus, "High quality time-scale modification for speech," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. New York, NY: IEEE, 1985, pp. 493–496.

[18] T. Dingler, J. Lindsay, and B. N. Walker, "Learnability of sound cues for environmental features: Auditory icons, earcons, spearcons, and speech," in *Proceedings of the 15th International Conference on Auditory Display*, Paris, France, 2008.

[19] D. Palladino and B. N. Walker, "Learning rates for auditory menus enhanced with spearcons versus earcons," in *Proceedings of the 14th International Conference on Auditory Display*, Montreal, Canada, 2007, pp. 274–279.

[20] ——, "Navigation efficiency of two dimensional auditory menus using spearcon enhancements," in *Annual Meeting of the Human Factors and Ergonomics Society*, New York, NY, Sept. 2008, pp. 1262–1266.

[21] M. Jeon, S. Gupta, B. K. Davison, and B. N. Walker, "Auditory menus are not just spoken visual menus: A case study of "unavailable" menu items," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI Work in Progress)*. Atlanta, GA: ACM Press, 2010, p. in press.

[22] L. Barkhuus and A. Dey, "Is context-aware computing taking control away from the user? three levels of interactivity examined," in *Proceedings of Ubicomp 2003*, 2003, pp. 149–156.

[23] K. Cheverst, K. Mitchell, and N. Davies, "Exploring context-aware information push," in *Personal and Ubiquitous Computing*, 2002, pp. 276–281.

[24] A. Howes, "A model of the acquisition of menu knowledge by exploration," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. Boston, MA: ACM Press, 1994, pp. 445–451.

[25] J. I. Kiger, "The depth/breadth trade-off in the design of menu-driven user interfaces," *International Journal of Man-Machine Studies*, vol. 20, pp. 201–213, 1984.

[26] D. P. Miller, "Depth/breadth tradeoff in hierarchical computer menus," in *Proceedings of the Human Factors Society Meeting*, Rochester, NY, USA, Oct. 1981.

[27] P. M. Commarford, J. R. Lewis, J. A. Smither, and M. D. Gentzler, "A comparison of broad versus deep auditory menu structures," *Human Factors*, vol. 50, no. 1, pp. 77–89, 2008.

[28] E. Mynatt and W. Edwards, "Mapping GUIs to auditory interfaces," in *5th Annual ACM Symposium on User Interface Software and Technology*. Monteray, California, United States: ACM, 1992, pp. 61–70.

[29] S. Brewster, "A sonically enhanced interface toolkit," in *Proceedings of the 3rd International Conference on Auditory Display*, Palo Alto, CA, U.S., 1996.

[30] B. K. Davison and B. N. Walker, "AudioPlusWidgets: bringing sound to software widgets and interface components," in *Proceedings of the 14th International Conference on Auditory Display*, Paris, France, 2008.