The Design of Sonically-Enhanced Widgets

Stephen Brewster
Glasgow Interactive Systems Group
Department of Computing Science
University of Glasgow, Glasgow, G12 8QQ, UK
Tel: +44 (0)141 330 4966, Fax: +44 (0)141 330 4913
Email: stephen@dcs.gla.ac.uk, Web: www.dcs.gla.ac.uk/~stephen/

This paper describes the design of user-interface widgets that include non-speech sound. Previous research has shown that the addition of sound can improve the usability of human-computer interfaces. However, there is little research to show where the best places are to add sound to improve usability. The approach described here is to integrate sound into widgets, the basic components of the human-computer interface. An overall structure for the integration of sound is presented. There are many problems with current graphical widgets and many of these are difficult to correct by using more graphics. This paper presents many of the standard graphical widgets and describes how sound can be added. It describes in detail the usability problems with the widgets and then the non-speech sounds to overcome them. The non-speech sounds used are earcons. These sonically-enhanced widgets allow designers who are not sound-experts to create interfaces that effectively improve usability and have coherent and consistent sounds.

Keywords: Earcons, auditory interfaces, widgets, non-speech audio, interface sonification, interface toolkits.

Introduction

With increasing amounts of information to take in when interacting with computers, users can become overloaded. What causes this problem? In our everyday lives we are able to deal with an enormous amount of complex information of many different types without difficulty. One reason for the problem is that computers communicate solely by graphical output, putting a heavy burden on our visual sense which may become overloaded. In the real world we have five sensory modalities and the combination of these avoids any one sense becoming overloaded. The next step forward in human-computer interaction is to use these other senses. Such multimodal interfaces would allow a richer and more natural communication between the computer and the user. They also allow the user to employ appropriate sensory modalities to solve a problem, rather than just using one modality (usually vision) to solve all problems (Brewster, 1997, Brewster *et al.*, 1995a).

Most current human-computer interfaces use just graphical output because the components, or *widgets* (for example, buttons, windows), they are built from were developed in the past when only graphics was available. Now more sophisticated output techniques are possible that allow more senses to be used and research is needed to investigate how these can be incorporated in to future human-computer interfaces to avoid the problems of overload and improve usability.

This paper suggests the use of non-speech sound output to enhance the graphical display of information at the human-computer interface. There is a small, but growing, body of research which indicates that the addition of non-speech sounds to human-computer interfaces can improve performance and increase usability (Brewster, 1994, Brown *et al.*, 1989, Gaver *et al*., 1991, Perrott *et al*., 1991). Non-speech sound is an important means of communication in the everyday world and the benefits it offers should be taken advantage of at the interface. Our visual and auditory senses work well together: The visual sense gives us detailed data about a small area of focus whereas the auditory provides data from all around. Users can be informed of important events even if they are not looking at the right position on the display (or even not looking at the display at all). This is particularly important for large-screen, high-resolution, multiple monitor interfaces. However, because this area is still in its infancy, there is little systematic research to demonstrate the best ways of combining these different media (Brewster, 1994).

Because this area is still in its infancy, sounds are often added in *ad hoc* ways by individual designers and this can lead to them being ineffective (Barfield *et al.*, 1991, Portigal, 1994). Arons & Mynatt (1994) suggest one reason for this: "…the lack of design guidelines that are common for the creation of graphical interfaces has plagued interfaces designers who want to effectively build on previous research in auditory interfaces". The aim of the research described here is to help designers to create effective sonically-enhanced interfaces. This paper describes the development of a sonically-enhanced set of widgets that provides for the use of sound in an effective and consistent way across the human-computer interface. It brings together previously published sonically-enhanced widgets, developments of earlier widgets and completely new widgets. It describes the overall structure of the sounds and how the widgets are linked together.

Aims of the sonically-enhanced widgets

There are three main aims for the sonically-enhanced widgets:

- *To ensure that the sounds added are effective and improve usability.* The sounds added are not gimmicks. Detailed investigations of usability problems have shown that sounds can be beneficial (Brewster, 1994).

- *To allow designers who are not sound experts to create sonically-enhanced interfaces*. Interface designers are often unskilled in sound design. A set of widgets that has the sounds included would remove the need for detailed knowledge of sound design. This follows the same approach as graphical interface toolkits (Myers, 1991) in that an interface designer without a detailed knowledge of graphic design can create an interface using a standard graphical interface toolkit. It is still possible to create interfaces that look bad but the possibilities are reduced by using a set of widgets that look and behave consistently. If designers use the widgets described here they will greatly increase their chances of making interfaces that sound good.

- *To make sure the sounds are used in a clear, coherent and consistent way across the interface.* This consistency avoids the problems of each application having its own sounds that mean different things in other applications. In graphical interfaces, widgets look consistent across different applications, e.g. a scrollbar looks the same in any application. By using the guidelines designers can ensure their widgets sound consistent across different applications.

The motivation for this research is that users' eyes cannot do everything. As mentioned, the visual sense has a small area of high acuity. In highly complex graphical displays users must concentrate on one part of the display to perceive the graphical feedback, so that feedback from another part may be missed. This becomes very important in situations where users must notice and deal with large amounts of dynamic data. For example, imagine you are working on your computer writing a report and are monitoring several on-going tasks such as a compilation, a print job and downloading files over the Internet. The word-processing task will take up all of your visual attention because you must concentrate on what you are writing. In order to check when your printout is done, the compilation has finished or the files have downloaded, you must move your visual attention away from the report and look at these other tasks. This causes the interface to intrude into the task you are trying to perform. It is suggested here that some information should be presented in sound. This would allow you to continue looking at the report but to hear information on the other tasks that would otherwise not be seen (or would not be seen unless you moved your visual attention away from the area of interest, so interrupting the task you are trying to perform). Sound and graphics will be used together to exploit the advantages of each. In the above example, you could be looking at the report you are typing but hear progress information on the other tasks in sound. To find out how the file download was progressing you could just listen to the download sound without moving your visual attention from the writing task.

Sounds used

The non-speech sounds used in the widgets are based around structured audio messages called *Earcons* (Blattner *et al.*, 1989, Brewster, 1994, Sumikawa *et al.*, 1986). Earcons are abstract, synthetic tones that can be used in structured

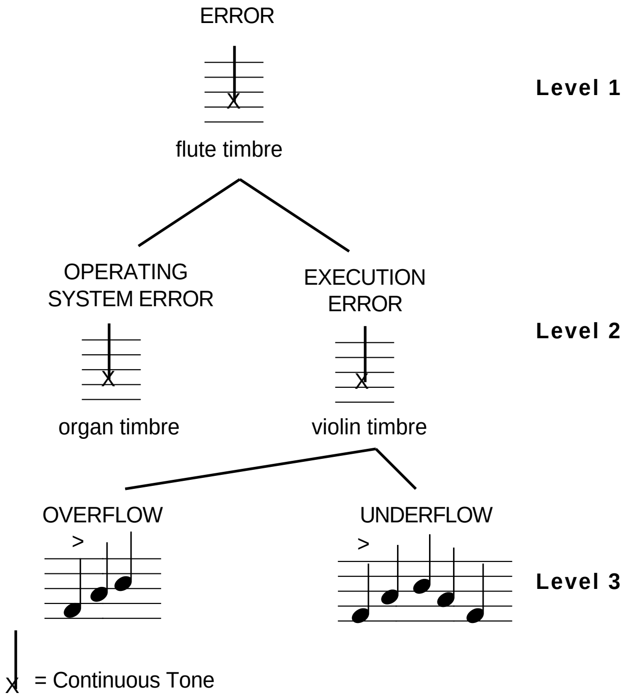

Figure 1. A simple hierarchy of earcons representing errors.

combinations to create sound messages to represent parts of an interface. Detailed investigations of earcons by Brewster, Wright and Edwards (1993) showed that they are an effective means of communicating information in sound.

Earcons are constructed from motives. These are short rhythmic sequences that can be combined in different ways. The simplest method of combination is concatenation to produce *compound earcons*. By using more complex manipulations of the parameters of sound (for example, timbre, pitch, rhythm) *hierarchical earcons* can be created (Blattner *et al.*, 1989) which allow the representation of hierarchical structures.

Figure 1 shows, as an example, a simple hierarchy of earcons based on one possible family of errors. Each earcon is a node on a tree and inherits the properties of the earcon above it. The different levels are created by manipulating the parameters of sound (for example, rhythm, pitch, timbre). In the diagram the top level of the tree is a neutral earcon. It has a flute timbre. The structure of the earcon from level one is inherited by level two and then changed. At level two there is still a continuous flute sound but new timbres are added to play alongside it. At level three a rhythm is added to the earcon from level two to create a sound for a particular error. This rhythm is based on the timbre from the level above. In the case of the overflow error there would be a continuous flute sound with a three note rhythm played on an organ accompanying it. Other levels can be created by using other parameters such as tempo, effects or spatialised three dimensional sound (Wenzel, 1992).

Using earcons, this hierarchy is easily extensible. For example, to add another major category of errors all that is needed is a new timbre. To create a new type of execution error only a new rhythm is needed and it can be added to the existing hierarchy. Therefore earcons provide a very flexible system for representing structured information.

Another method of presenting information in non-speech sound is *Auditory Icons* (Gaver, 1989). These use natural, everyday sounds, instead of musical ones, which have an intuitive link to the action or object they represent in the computer interface. It is claimed that this makes them very easy to learn as users can use their existing knowledge. One problem with auditory icons is that they cannot easily be used to build up hierarchies of sounds. As will be described below, the relationships between the sounds in the widgets are an important part of the design and it is much easier to do this with earcons. With auditory icons it can also be difficult to get a natural sound that has an intuitive link to an interface object. For example, what would be the natural equivalent of a listbox? With earcons an abstract musical sound can be constructed to represent anything; the user then has to learn the mapping. Results from Brewster *et al.* (1996) have shown that these mappings are easy to learn. It was for these reasons that earcons were used for the sonically-enhanced widgets.

Widgets and toolkits

Before the sonically-enhanced widgets are described standard graphical widgets and toolkits will be discussed. Myers (1991, p 14) defines a toolkit thus: “A toolkit is a library of interaction techniques that can be called by application programs”. Therefore the toolkit provides a set of ready-made interaction techniques, or widgets, that developers can use in their applications. Widgets are now common across almost all graphical human-computer interfaces (Myers, 1990). As Myers (1991, p 17) says: “Although there are many small differences among the various toolkits, much remains the same. For example, all have some type of menu, button, scroll bar, text input field, etc.”. Toolkits are one of the most common methods of creating graphical human-computer interfaces.

One advantage of toolkits is that they can save the programmer much effort when developing interactive systems. The widgets in a toolkit have a pre-defined set of behaviours that the developer gets for free (without any further programming effort) by using the toolkit (Dix *et al*., 1993). The toolkit enforces consistency between applications that use it which is good for the user because interfaces work in a predictable way, and is good for the developer because he/she saves programming effort and gets a consistent look-and-feel. For these reasons it was decided that a set of sonically-enhanced widgets would be developed.

A set of standard widgets will now be discussed to show their functionality and to provide background to the sonic-enhancements described below. Widgets from the Tk toolkit for X Windows will be used as an example (Ousterhout, 1994). More detailed descriptions of interactions with each type of widget is given in the appropriate part of the section below entitled ‘Overall structure of the sonically-enhanced widgets’.

Frames: These are the simplest widgets and are just coloured regions (perhaps with a three-dimensional border). These are used as containers to group other widgets. They do not normally respond to mouse or keyboard events, they are for controlling display of other widgets. Frames may have layout policies that define how the widgets they contain are to be displayed on-screen. For example if the layout policy is horizontal all the widgets within the frame will be laid out horizontally across the screen.

Buttons: These are one of the most common widgets in a graphical interface. They are similar to labels but respond to mouse events so that the user can press them. Buttons in some systems will highlight as the mouse is moved over them to indicate that they can be pressed. If the button is pressed it will highlight (for more details on interacting with buttons see the section ‘sonically-enhanced buttons’ below). In addition to this type of button there are also checkboxes and radio buttons. Checkboxes are used for making binary choices and radio buttons allow the selection of one item from a mutually-exclusive set of choices.

Dialogue and Alert Boxes: These are information windows that are used by the system to bring the user’s attention to some important information, perhaps an error or warning. They are also used to invoke a sub-dialogue for a specific task within a larger task. For example, editing a document might be the larger task with the sub-task of setting font options done in a dialogue box.

Listboxes: These widgets display a collection of items (e.g. strings) and allow the user to select one. If there are more items than can fit in a window the user can scroll to reach the others. The user usually selects an item in the list by clicking the mouse.

Scrollbars: Scrollbars are associated with another widget and control the view in that widget (for example, a listbox). Scrollbars usually display an arrow at each end (for line by line scrolling) and a slider (or thumbwheel) for scrolling by larger amounts. The user can also click either side of the slider to scroll by one window full. Scrollbars can also be used for scales (or valuators). These allow the user to change the value of a numeric field by dragging the slider.

Menus: There are several different types of menu but all of them allow a user to choose from a set of items. The menu is normally hidden and can be made visible when needed by the user (for example, by clicking on a menu bar). One of the most common types is the pull-down menu. The user clicks on the title of the menu in the menu bar and the menu is displayed. The user can then move the mouse over the required item to select it. Associated with the pull down menu may be cascaded, or hierarchical, menus. When the user moves the mouse over a menu item in a menu, a further menu may be displayed to the right. This allows more menu items, associated with the first, to be displayed. Pop-up menus are similar to pull-down menus but can appear in places other than the menu bar.

The widgets in an application form a hierarchy. The application might bring up a window as in Figure 2. This window is made up from a frame which contains a menu bar, a listbox and a scrollbar. The menu bar contains two menus and, in turn, these menus will contain menu items. This hierarchical structure was used to define the sounds for an application.

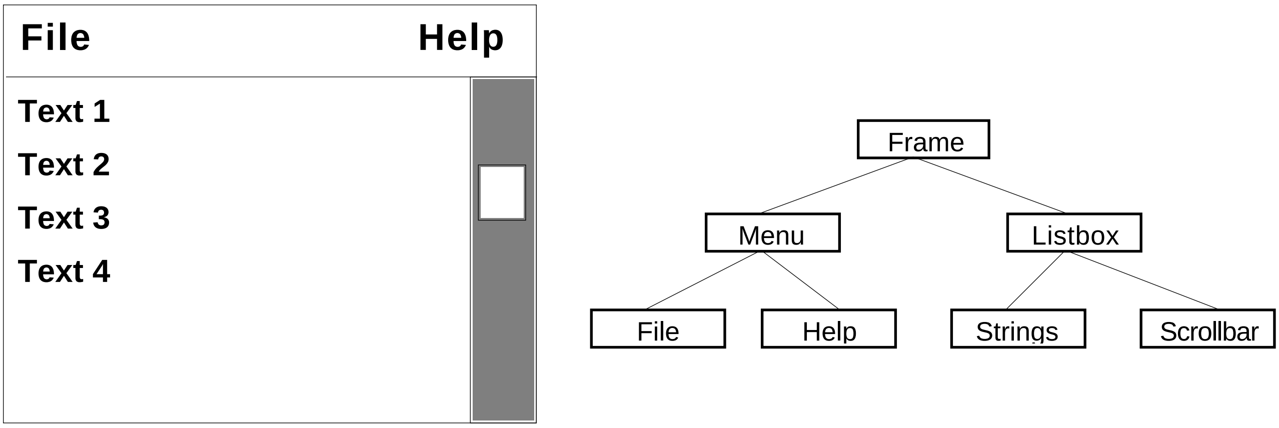

Figure 2. A simple window with several widgets and the hierarchy used to represent them.

Overall structure of the sonically-enhanced widgets

The overall structure of the earcons in the widgets was as follows: Each application had its own timbre as a base for all of its sounds (just as an application has a base frame for all of its widgets). All widgets within an application used this and then modified it by changing the rhythm, pitch, etc. Figure 3 shows such a hierarchy. At level 1, the three applications all have different timbres. These are inherited by level 2 and modified with pitch, rhythm, etc. These modifications are constant across applications so that widgets in different applications sound consistent (similar to graphical widgets that look consistent across applications). This approach to allocating earcons is the same as for graphical widgets: A graphical application has a base frame for its widgets. This frame has a spatial location and any widget drawn within the frame uses the base spatial location to position itself.

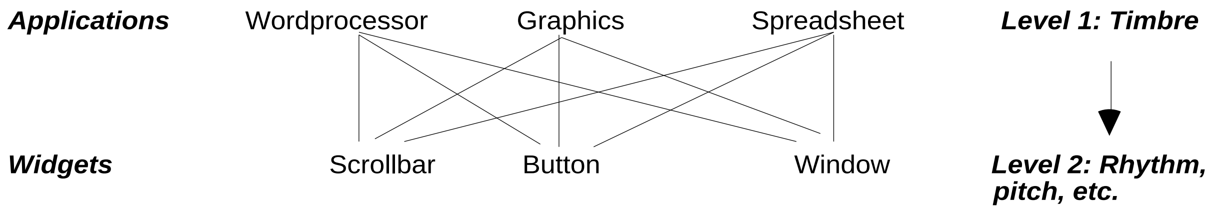

Figure 3. A hierarchy of sonically-enhanced widgets across applications.

As an example, consider a button widget in the three applications in Figure 3. If a button was used in the Wordprocessor application it would have the Wordprocessor instrument, for example an organ. The button would have its own rhythm and note structure, for example a two note chord. In the Wordprocessor application this chord would be played by an organ. If the Spreadsheet application had a piano timbre then the same two note chord would be played but modified by that instrument. In this way the earcons for each widget would be consistent across the whole interface (the button would always use the same two note chord) but would also fit with the sounds of the application of which it was part. This is the same as a button in a graphical interface which always looks the same but is spatially related to the application to which it belongs (by appearing within the bounds of the applications' frame).

In practice, it may not be possible to give all applications a different timbre. Instead, applications could be grouped (for example, communications, text processing or graphics) and each group given a different timbre. Different applications within a group could then be separated by register (register is a musical term indicating position in a musical scale; a note in a high register would have a high pitch).

Sonically-enhanced widgets

Earcons were used to overcome usability problems in standard graphical widgets. Each of the widgets in a standard toolkit has been analysed to discover any usability problems. This was done using a structured method proposed by Brewster and colleagues (Brewster, 1994, Brewster *et al.*, 1994) that analyses interactions in terms of event, status and mode information to find where usability problems may occur due to incorrect feedback. From this analysis earcons were created for the auditory feedback. Many of these widgets have been tested and the results show significant improvements in usability (for more details see the section "Results of sonic enhancement" below).

The new widgets use sound to enhance the graphical display of information. These widgets are not aimed at blind computer users who would need a complete representation in sound. Much work in that area has been done by Mynatt *et al.* (Mynatt, 1992, Mynatt and Weber, 1994) and Savidis & Stephanidis (1995), amongst others. The aim of the widgets described here is to provide feedback that might otherwise be missed by sighted users. This will also provide an advantage to partially-sighted computer users. They often have similar, but more pronounced problems than sighted users when it comes to missing feedback because of looking in the wrong place. The often greatly reduced area of focus for partially-sighted users makes it very easy for them to miss feedback. The enhancements to graphical widgets described here will help them.

The earcons for all of the widgets were designed using the earcon guidelines proposed by Brewster *et al.* (1995b). In addition, they were also designed to be as short and simple as possible. The sounds had to be able to keep pace with the visual interactions they accompanied. If the sounds took too long to play then they would either hold up the interaction or refer to an interaction that had already completed, neither of which was acceptable. Short sounds would also minimise annoyance; Potential users of sonically-enhanced interfaces are often worried that the sounds will become annoying. One other principle used when designing the sounds was to keep them as quiet and short as possible to avoid annoyance due to sound pollution (Berglund *et al.*, 1990). Results of experimental investigations of several of the widgets showed no increase in annoyance (Brewster, 1994).

The following sections describe some of the sonically-enhanced widgets in the toolkit. For each of the widgets, the usability problem to be corrected will be described and then the sonic enhancements discussed. This paper will just discuss the design of the widgets, for more details on the experiments to test their usability the reader is referred to the references given in the appropriate section. Some widgets have been published elsewhere in less detail (buttons), some are developments of earlier widgets (scrollbars) and others are completely new. The widgets will be described below as if they were part of an application that used an electric organ timbre as the base for its sounds.

Sonically-enhanced buttons

One of the most fundamental widgets in all graphical human-computer interfaces is the graphical button (to avoid confusion, *graphical button* will here be used to refer to the button on the computer display and *mouse button* will be used to refer to the button on the mouse). Although these are very common they are not without problems (Dix *et al.*, 1993, Dix and Brewster, 1994). One of the main difficulties is that the user may think the graphical button has been pressed when it has not. This can happen because the user moves off the graphical button before the mouse button is released. This is caused by a problem with the feedback from the graphical button (see Figure 4). Both correct and incorrect presses start in the same way (1A and 2A). In the correct case, the user presses the graphical button and it becomes highlighted (1B), the mouse button is then released with the mouse still over the graphical button, it becomes un-highlighted (1C) and the operation requested takes place. The button slip-off starts in the same way. The user presses the mouse button over the graphical button (2B), then moves (or slips) off the graphical button and releases the mouse button (2C), the graphical button becomes un-highlighted (as before) but no action takes place. The feedback from these two different situations is *identical*. This problem occurs infrequently but, as the error may not be noticed for a considerable time, the effects can be serious. With a one-step undo facility users must notice before the next action takes place otherwise they may not easily be able to correct the mistake.

The identical feedback would not be a problem if the user was looking at the graphical button to see the slip-off, but this is not the case (Dix and Brewster, 1994). The error is an example of an *action slip* (Reason, 1990). These occur with expert users who perform many simple operations (such as button clicks and menu selections) automatically and do not explicitly monitor the feedback from each interaction. Lee (1992) describes such errors thus (p 73): "…as a skill develops, performance shifts from 'closed loop' control to 'open-loop' control, or from monitored mode to an automatic, unmonitored mode of processing." As users become familiar with a simple task they no longer monitor the feedback from it so closely. In this case the automatic task is the button click (which most users are very familiar with). The user will be concentrating more on the main task he/she is trying to perform than on the click. Many of the errors described in the widgets below are action slips; users do not monitor the feedback from the widget but continue to look

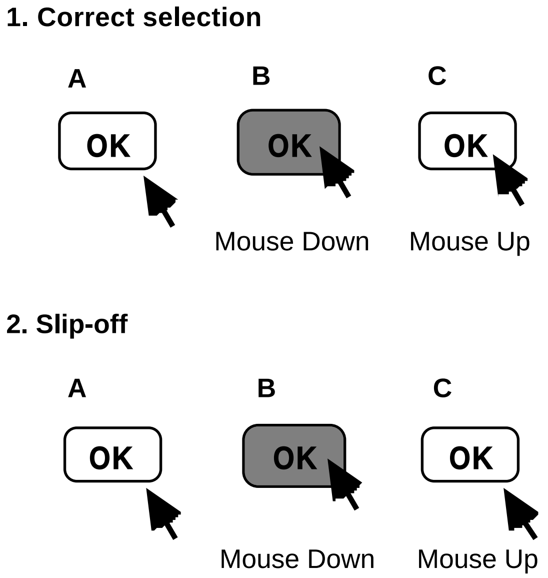

Figure 4. Feedback from pressing and releasing a graphical button.

at the information in which they are interested. If they must look at the widget then it forces the interface to intrude upon the task they are trying to perform.

One problem that compounds the difficulties of actions slips is *closure* (Dix *et al.*, 1993). Closure occurs when a user perceives a task as being completed. In some cases the task may appear to be completed when it is not. The user may experience closure and carry on to do something else and so cause an error (Brewster *et al.*, 1995a, Dix and Brewster, 1994). Dix & Brewster (1994) suggest there are three conditions necessary for such slip-off errors to occur:

i) The user reaches closure after the mouse button is depressed and the graphical button has highlighted.
ii) The visual focus of the next action is at some distance from the graphical button.
iii) The cursor is required at the new focus.

In this case, closure is reached when the graphical button is highlighted (the mouse button is down). In reality, the task does not end until the mouse button is released. Because closure (i) is reached before this, the user starts the next task (mouse movement, iii) in parallel with the mouse up and a slip-off occurs. The user’s attention is no longer at the graphical button (ii) so the feedback indicating a slip-off is not noticed.

These problems occur in graphical buttons that allow a ‘back-out’ option: Where the user can move off the graphical button to stop the action. If the action is invoked when the mouse button is pressed down on the graphical button (instead of when it is released) then these problems do not occur as the user cannot slip off. These buttons are less common because they are more dangerous as users cannot change their minds.

In this situation sound should be used because the user’s eyes are occupied. Moving the mouse to the location of the next action requires visual attention so that the mouse can be positioned correctly. Therefore, the user cannot look at the button to see feedback indicating a slip-off. It would be very difficult to correct this problem using visual feedback. Buttons could be changed so that they indicated a difference between a successful and unsuccessful click. For example, the button could flash in a different way. This would not work because users will not be looking at the button but at the location of the next action. The area of visual focus is too small to allow them to see the feedback. Feedback could be given at the mouse location but again we cannot be sure that the user will be looking there either. Sound would allow us to present the information to the user without knowing where he/she was looking.

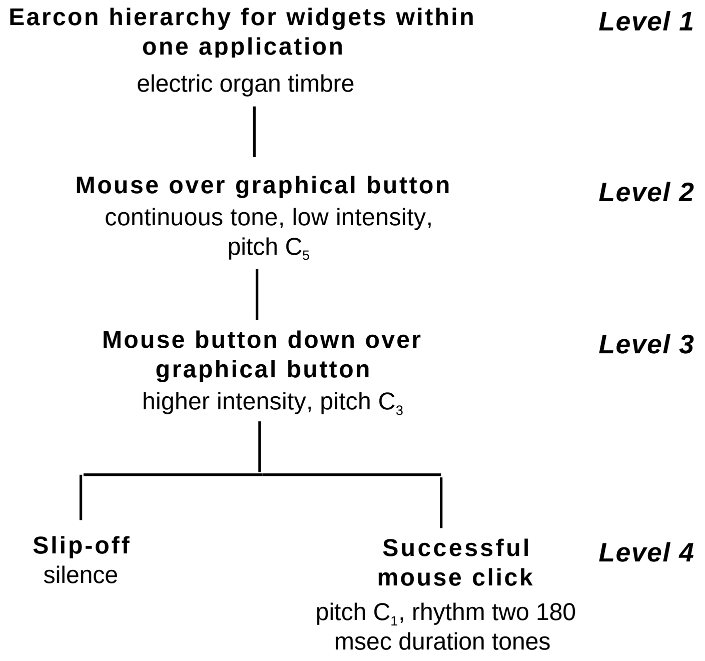
Earcon hierarchy for widgets within one application
electric organ timbre
Level 1
Mouse over graphical button
continuous tone, low intensity, pitch C_5
Level 2
Mouse button down over graphical button
higher intensity, pitch C_3
Level 3
Slip-off
silence
Successful mouse click
pitch C_1, rhythm two 180 msec duration tones
Level 4

Figure 5. The hierarchy of earcons used in the sonically-enhanced buttons.

The design of the sonically-enhanced buttons

Three sounds were needed to overcome the usability problems discussed above: One to indicate to the user when the mouse was over a graphical button; one to be the auditory equivalent of the graphical highlight when the mouse button was pressed down on the graphical button; the other to indicate when a button was pressed correctly or when a slip-off occurred. Figure 5 shows the hierarchy of earcons used. The top of the hierarchy shows the electric organ instrument to be used for the application. Each level inherits properties from the levels above. For example, at level 3 (mouse down over graphical button) the timbre is inherited from level 1 and the continuous tone from level 2. At this level, the new intensity and pitch overwrite those from level 2. For details on the experimental evaluation see (Brewster *et al.*, 1995a).

A base sound was created for when the mouse was moved over a screen button. This was a continuous tone at C_4 (130Hz). The volume was kept to just above the threshold level. This sound was played as long as the mouse was over a graphical button; it stopped when the mouse was moved off. The sound was quiet and low-pitched to avoid annoyance because users frequently move the mouse over buttons. When the mouse button was pressed down over a graphical button a continuous sound at pitch C_3 (Middle C, 261Hz) was played. This was a more attention grabbing sound than the previous one to indicate that an interaction was taking place. This continued for as long as the mouse button was down and the mouse was over the graphical button. If the mouse was moved off the graphical button the sound stopped. If the mouse was released over the graphical button then a success sound was played. This consisted of two notes, played consecutively, at C_1 (1046Hz) each with a duration of 40 msec. This success sound had to be kept short so that users would not get confused as to which button the feedback was coming from: The audio feedback had to be able to keep pace with interactions taking place. It also had to be perceivable - if it was too short then users would not be able to hear it (Moore, 1997). The duration chosen met these criteria. To make sure that the number of sounds was kept to a minimum and speed maximised, if a user quickly clicked the mouse over a graphical button only the success sound was played. The mouse button down and success sounds differentiated a successful and unsuccessful mouse click.

The earcons used a combination of pitch, duration and intensity to get the listener's attention. This meant that a lower level of intensity could be used, making the sounds less annoying for the primary user and others working nearby. Annoyance is most often caused by excessive intensity (Berglund *et al.*, 1990). It is important to note that intensity is not the only way to get the user's attention in sound. The human perceptual system is good at detecting dynamic stimuli. As Edworthy *et al.* (1989) showed, attention-grabbing sounds can be created by varying other sound parameters, such as those used here.

Sounds are made only when feedback must be presented to the user. This is different to the approach taken by Pitt and colleagues (Pitt and Edwards, 1991, Pitt and Edwards, 1995), for example. In their systems sound was continuously emitted to indicate proximity to icons and buttons. The difference is that their systems were aimed at blind users who needed information as to the location of targets in the interface. As mentioned, the approach here was to aid sighted users. The use of continuous sound has the disadvantage of annoyance whereas the use of sound only at specific interaction points avoids this.

Sonically-enhanced menus

Menus are very similar to buttons. Menu items are selected by moving up or down a menu with the mouse button pressed down and then releasing the button over the required item. Users can back-out if they want to by moving off the menu but this facility can again can lead to slip-off errors. In some systems (such as the Macintosh) the menu can be made to flash if a correct selection is made. This is different to graphical buttons in that there is a difference in the feedback between a correct and incorrect menu selection. However, as mentioned above, it may still not help as the user will be looking at the location of the next action rather than the menu. The narrow focus of visual attention then prevents him/her from seeing the feedback.

There is another problem that menus have which buttons do not. In a menu the user can slip-off the required item on to one above or below. This will still be presented as a correct selection (the user chose an item from the menu correctly) but will not be the item required. If the user slipped off when trying to save and did not notice, the data would not be saved with perhaps serious consequences.

The problems are again due to action slips and closure. Menu selections are simple, automatic actions for experts so feedback is not monitored closely and will not be noticed. The user will perceive a menu item as selected (and so reach closure) when the menu item highlights, it is not actually selected until the user releases the mouse, so errors occur. This can happen, in a similar way to that described for buttons, because the user moves the mouse from the menu to the location of the next interaction. The movement overlaps with the release of the mouse button so that the user releases in the wrong menu item. The user will be about to release on the item required, will start to move the mouse to the location of the next action and so slip into one of the other menu items and then release.

An example of this error consider selecting ‘Save’ from the file menu in a word processor. The user will be more concerned with the document being saved than the mechanics of the menu, making action slips likely. He/she would be typing in the document and decide to save. The mouse (and visual attention) would be moved up to the menu to select save, the menu item would be highlighted, the user would see the highlight, reach closure and then begin to move back to the document to continue typing. If this return movement is done too soon, the mouse may slip on to another item or out of the menu before the interaction is complete. In many systems no feedback is given to indicate a save has been performed, so the user would have no indication of his/her error.

In order to solve the problems users must perceive the feedback from the menu. Therefore the right feedback must be given to ensure users know what is going on (Reason, 1990). The Macintosh-style multiple flash goes some way to dealing with this problem, but is not a complete solution because any feedback from the menu requires the user's visual attention to be on the menu and this will not be the case. It is again difficult to solve these problems with extra graphical feedback. Graphics displayed on the menu will not be seen by users because their attention will have moved back to the main task they were engaged in. Information could be displayed at the mouse location but we do not know if users are looking there either. In this case, non-speech sound has certain advantages. It can be heard from all around, it is good at getting our attention whilst we are looking at something else and it does not disrupt our visual attention. If menu information is given in sound then there is no need to know where users are looking. If users must look at the menu then it forces them to stop what they are doing for their main task and causes the interface to intrude upon the task they are trying to perform, sound does not have such drawbacks.

Other attempts to solve the problems of menus

There has been some previous work on adding sound to menus, although to solve different problems. Karshmer *et al.* (1994) added sounds to a menu system to aid navigation for blind users. They were not trying to overcome the slip off problems but to aid users in navigating around menus structures. The navigation was implemented by either changing the tones and timbres of the items in the menus, or by using synthetic speech to tell users their position in the menus. They also dealt with the problems of hierarchical menus, which can be difficult for blind users. Navigation through menus for blind users was not the focus of the work here, its aim was to overcome slip-off problems in graphical menus for sighted users, that is why a different approach was taken. Hierarchical menus have exactly the same slip-off problems as ordinary menus so the solutions suggested here will work for those too.

Barfield *et al.* (1991) carried out similar experiments where they used earcons to aid navigation through a menu hierarchy. The earcons they used were very simple, just decreasing in pitch as a participant moved down the hierarchy. The sounds lasted half a second. They describe them thus (p 104): "…the tones were played with a harpsichord sound and started in the fifth octave of E corresponding to the main or top level of the menu and descended through B of the fourth octave". These sounds did not fully exploit all the advantages offered by earcons (for example, they used neither rhythm nor timbre and did not exploit the highly structured nature of earcons) and did not improve user performance in the experimental task. Using pitch alone to differentiate the items was shown to be ineffective in the experiments of Brewster *et al* (Brewster, 1994, Brewster *et al.*, 1992, Brewster *et al.*, 1993). If better earcons had been designed then advantages may have been found. Again, the aim of the work just described was navigation through menu structures. The sonically-enhanced widgets were aimed at sighted users who could see the graphics. The problems that they suffer from are slipping-off menu items, therefore that is why a different approach was taken.

The design of sonically-enhanced menus

The sounds were based around those of the sonically-enhanced buttons above, as the problems to be addressed were similar (for more details of the experimental evaluation see (Brewster and Crease, 1997)). Three earcons were needed to deal with the problems described:

1. An earcon was played when a menu was displayed. A family of organ sounds was used to indicate that the menus were related (part of the same application). The File menu had a percussive organ, the Parts menu a drawbar organ and the Destinations menu a rock organ. These sounded related but distinguishable so that the user could tell which menu was making the sound. A low intensity, continuous note at pitch C_3 (523 Hz) was played for each of the menus. The

sound continued as long as the cursor was in the menu. If the user moved the cursor out of the menu the sound stopped (in the same way as the sonically-enhanced buttons).

2. To deal with item slips a combination of two earcons was used. A highlight sound was created that was similar to the standard graphical highlight (and the highlight sound used for the buttons above). This was again a continuous, low intensity tone which used the timbre of the menu that the cursor was in. The sound alternated in pitch from B_2 (987 Hz) for odd numbered items and E_3 (329 Hz) for even numbered items. So the first item in the menu would be played at E_3, the second at B_2 and the third a E_3. This sound started when the mouse had been over an item for half a second. Only two sounds were needed to indicate the movement from one menu item to another as slip-offs only occur to items directly above or below the current position. These pitches were chosen to make the two earcons sound distinct and recognisable. This sound was stopped if the user moved the mouse out of the menu or moved over a divider or disabled item.

3. The final earcon indicated a selection. This could be either a correct selection or an item slip. For a correct selection the earcon was based around the timbre of the menu the cursor was in and the pitch of the sound was based on the highlight sound for the menu item. Using these as a base, two 40 msec duration tones were played at a higher intensity. To indicate an incorrect selection the timbre of the current menu was again used. However, this time a fixed rhythm of three notes of 40 msec each at pitch C_2, B_2 then F_2 was played (these sounded discordant and therefore attention-grabbing, see Figure 6). This was not dependent on the highlight sound for the current menu item. This sound was always the same, indicating the item slip error in each of the different menus. If the user released the mouse over a divider then no sound was played.

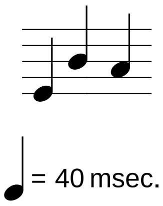

Figure 6. The error earcon indicating a miss-selection.

To avoid any potential annoyance due to the earcons we made sure they were all played at low volume. The menu earcon (1) indicated a menu slip by a lack of sound and the highlight earcon (2) didn’t start playing until the participant had been over an item for half a second. In a normal, fast interaction this sound was not played, the user only heard the menu sound (1) and the selection sound (3).

Sonically-enhanced scrollbars

There are several usability problems with scrollbars (Brewster *et al.*, 1994):

- *Position awareness in documents:* When scrolling through a document it can be hard to maintain a sense of position. The text can scroll by too fast to see (and the thumb wheel only gives an approximate position). The user really wants to look at the information in the window, not the scrollbar. However, in order to get location information he/she must look at the scrollbar, forcing visual attention away from the document which is what is really of interest and causing the interface to intrude into the task being performed. As before, the user cannot be looking in two places at once. Some systems, such as Microsoft Word, put a page count at the bottom left hand corner of the screen but this is too far from the centre of visual focus - again the user must move his/her visual focus to the page counter. Sound can be used to present this location information so that the user can perceive it wherever he/she is looking.

- *'Kangarooing' with the thumb wheel:* Repeatedly clicking in the scroll area above or below the thumb wheel scrolls by a window-sized step. Clicking below the thumb scrolls down in the document and clicking above scrolls up. Figure 7 shows an example of kangarooing. In A the user begins clicking to scroll down towards the mouse pointer. In B the thumb wheel is just above the pointer. In C the user has clicked and the thumb has scrolled below the pointer. In D the user clicked again and this time the thumb scrolled back above the pointer and kangarooing occurred. Unless he/she is looking at the thumb it can be hard to recognise that this has happened. If the user continues clicking the thumb wheel will bounce above and below the mouse pointer location. If clicking is rapid, this can be difficult to see because the information in the window is changing rapidly. Information about kangarooing errors could be presented in sound so that the user could tell if even if he/she is looking at the document.

- *Moving the mouse pointer outside the thumb wheel:* Dragging the thumb wheel allows the user to scroll by an arbitrary amount. When dragging in systems such as the Macintosh, the document does not scroll until the mouse button is released. When dragging the thumb up or down in the document one may move too far to either the top, bottom, left or right of the scroll bar (outside of the 'hotspot') and the thumb is dropped. The document then does not scroll when the mouse is released and the user may become confused. Often, only the outline of the thumbwheel is dragged, the real one stays where it is, and this does not grab the user's attention because it is harder to see. Dragging within the area of the scrollbar requires visual attention to be taken away from the document, if this information was presented in sound, users would be able to hear when they had moved outside of the scrollbar area.

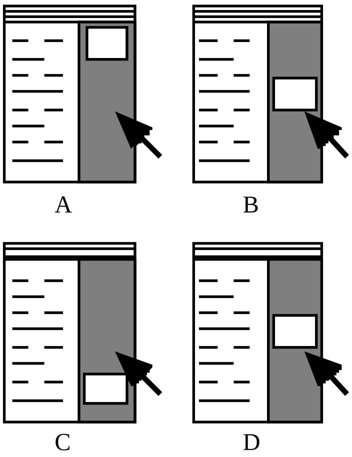

Figure 7. Scrollbar 'kangarooing'.

Other attempts to solve the problems of scrollbars

Beaudouin-Lafon & Conversy (1996) used a similar approach to give information to overcome kangarooing problems. They used an auditory illusion called Shepard-Risset tones which increase (or decrease) in pitch indefinitely (similar to the Escher drawing of an endless staircase). When the user was scrolling down a continuously decreasing tone was used when scrolling up an increasing one. If kangarooing occurred then the user would hear tones moving in the wrong direction. This is similar to the approach taken here. The tones used by Beaudouin-Lafon & Conversy do not permit the use of timbre - they are based around simple sine waves, the sounds are also hard to control to keep the illusion going. For these reasons they were not used here.

The design of the sonically-enhanced scrollbars

Two types of sounds were needed to solve the problems described above: One to give scrolling information to indicate when the thumbwheel reached the target location (to avoid kangarooing) and one to give location information. Figure 8 shows the hierarchy of earcons used. See (Brewster *et al.*, 1994) for details on the evaluation of an earlier, simplified version of the sonically-enhanced scrollbar.

The first sound was a fixed tone of duration 180 msec. and was used to indicate a window scroll event with the thumbwheel. The sound was kept short so that it could keep up with the interactions taking place. If the user scrolled

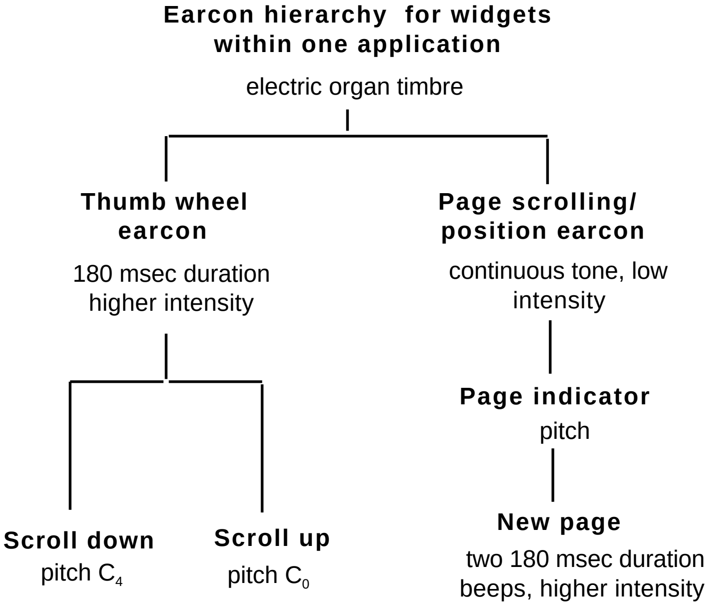
Earcon hierarchy for widgets within one application
electric organ timbre
Thumb wheel earcon
180 msec duration higher intensity
Page scrolling/ position earcon
continuous tone, low intensity
Page indicator
pitch
Scroll down
pitch C_4
Scroll up
pitch C_0
New page
two 180 msec duration beeps, higher intensity

Figure 8. The hierarchy of earcons used in the sonically-enhanced scrollbars.

towards the bottom of a document, the short tone was played at a low-pitch, C_4 (130Hz). When scrolling up a high-pitched note C_0 (2093Hz) was played. High pitch was used as up and low pitch as down because of the natural tendency to perceive higher pitch as higher spatial location (Mansur *et al*., 1985). If a user was clicking to scroll down towards the mouse pointer location he/she would hear the repeated low-pitched sound. If kangarooing occurred then the user would hear a demanding high-pitched tone when not expected and this would indicate the error.

A low intensity, continuous tone was used to give location information. This earcon changed in pitch when a page boundary was crossed; lower pitch when scrolling downwards and higher when scrolling upwards. To indicate a page boundary event the background tone was increased in volume for two tones of 180 msec. each to demand the listener’s attention. It then decreased again to just above threshold level so that it could be habituated. The different number of notes differentiated this earcon from the previous window scroll sound. Again the sound was short so that it did not hold up the interaction. The notes played when scrolling towards the bottom of the document decreased in pitch from B_1 (1975Hz) to C_4 (130Hz) when a page boundary was crossed. The reverse occurred when scrolling up from the bottom of the document. When the scrollbar was clicked the thumb sound was played first followed by the page boundary sound after a 180 msec. delay (if a page boundary had been crossed).

The continuous location sound was only played when the mouse pointer was in the scrollbar area. If the user moved the mouse out of the scrollbar then the sounds stopped. The continuous sound would become annoying if played all the time, so it was only played when the user was engaged in scrolling. This earcon was also used to stop the problem of moving the mouse pointer outside the thumb. When this happened the sounds stopped so that users could then tell (by the lack of sound) that they had slipped out of the scrollbar. When dragging the thumb the earcon changed in pitch as described above for the location indicator, changing when the user dragged over a page boundary. This gave location information when the user's eyes were busy looking at the document. The same earcon was used for scrolling with the arrow buttons at the top or bottom of the scrollbar (which allow line by line scrolling). The location sound played and changed when the user crossed a page boundary. If the user moved the mouse out of the arrow (with the mouse button down) then the sound stopped, indicating that the mouse had slipped off the button.

Sonically-enhanced alert boxes

Alert boxes are used to alert the user to errors or other important information, for example Figure 9. One problem is that the user may not notice them (e.g. a 'hunt-and-peck' typist who looks at the keyboard whilst typing or a touch-typist who is looking at a source document, may not notice a box in the screen) or they may be obscured by another window on the display. In many systems a simple error tone is played to indicate the appearance of an alert box. Unless this is very loud (and thus annoying) it can be missed. It is transient so the tone only lasts for a short time and then disappears. If a loud, constant sound was played then this would quickly become annoying. It also gives no indication of the type of error, the sound is the same for every error that occurs. One way to avoid these problems is to use better designed sounds. Patterson has designed 'ergonomic' alarms and warnings for aircraft (Patterson, 1982) and the principles he developed can be used to design the earcons for alert boxes.

Patterson's warnings are made up of smaller units called 'bursts' which are, in turn, made from 'pulses'. The whole warning repeats over a period of time. The repetition is fastest when the alarm first appears (to get the user's attention) but then repeats less frequently to remind the user to deal with the problem. This avoids the transient nature of current error sounds. The earcons used for alert boxes were based on the timbre for the application to which it belonged, for example an error in a compilation would use the base instrument of the compiler application. The first burst would be played when the alert box appeared, the second after 4 secs., then after 10 secs., then 15 secs. as defined by Patterson (1982). This allowed the earcons to gain the listener's attention without annoyance, and keep reminding him/her to deal with the alert box.

A pulse was a 200 msec tone played in the instrument of the application, at pitch C_3 and medium intensity. A burst was six of these short pulses lasting two seconds in total. The first two pulses were played at lower intensity, the second two

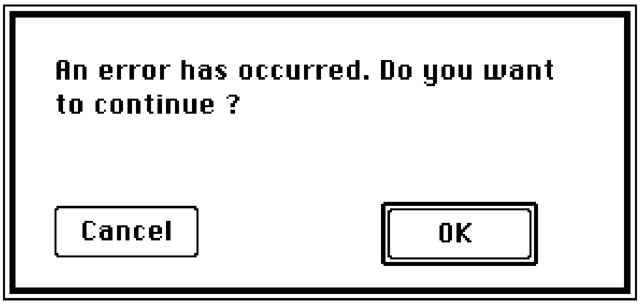
An error has occurred. Do you want to continue ?

Figure 9. An alert box.

played louder and the final two again at low intensity (see Figure 10). In this case the sound just indicated the appearance of the alert box. However, the rhythm of the pulses could be changed to indicate different errors, as in Figure 1. As soon as the user dismissed the dialogue box the sounds stopped.

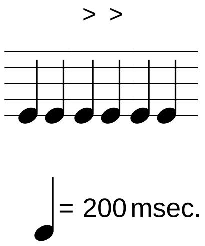

Figure 10. A burst made up of six pulses. Each pulse lasted 200 msec. with a gap of 160 msec between each pulse. The central pair of pulses was played louder than the first and last pairs (see Patterson, 1982 for more details).

Sonically-enhanced windows

One common error in multiple window environments is the ‘unselected-window’ error (Brewster, 1994). This occurs when the user tries to interact in one window but another is actually the selected, active one. This may cause keystrokes to be lost or errors to occur because the selected window is interpreting commands not meant for it (Harrison and Barnard, 1993). There are two situations where this may occur (Harrison and Barnard, 1993):

- On resumption after a pause the user’s focus has changed so that he/she is no longer ready to engage in the interaction the system is expecting. For example, coming back to the system after having been away for some time, the user must re-orient to the system. Often the first task to be done will cause the user to re-orient to that window because it is a priority, whereas another window is really the active one.

- Discontinuity of input, for example when the mouse is knocked into another application’s window by mistake (in systems where the mouse need only be placed in a window for that window to become the active one). The user

> still thinks the old window is active but it is not. A further example is when a user activates another window to look at some data, but then fails to re-activate the main window again when switching attention back to it.

The way to solve this problem is to reinforce the feedback from the active window so that the user does not mistake it. As Reichman says (1986, p 296): “Most of these systems mark the active window by a little blinking pointer or a highlighted title bar. These visual indicators are too limited”. The user is likely to be looking at the contents of a window rather than the title bar. This means that it is outside the area of visual focus so that the user will not see the grey indicating an inactive window. Using sound, information about the active window can be communicated to the user wherever he/she is looking.

The design of the sonically-enhanced windows

Figure 11 shows the hierarchy of sounds used in the sonically-enhanced windows. The auditory feedback to reinforce the active window on resumption after a pause came partly from the other widgets described above. If users pressed a button, used a menu or scrollbar, they heard the timbre of the currently active application. This may have been the sound of the application they were expecting (so there was no error) or it may have been the sound of an application they were not expecting, so they would notice that an unselected window error had been made. This makes use of the hierarchical nature of the sounds used and the hierarchical nature of widgets (see Figure 2). As the earcons of the widgets in the hierarchy were based on the base sound for the application they could indicate to users which application was being used. Using sound in this way does not rely on users noticing the colour of the title bar of the window, the sounds could be heard wherever they were looking in the window.

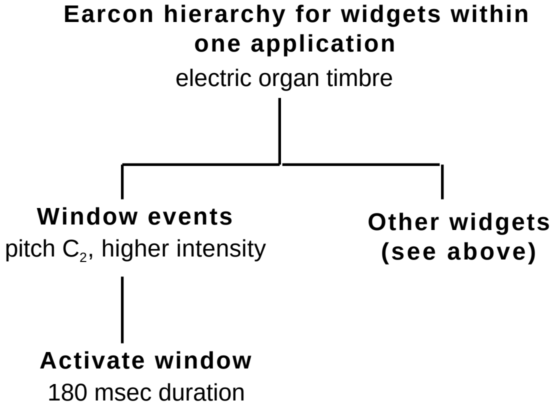
Earcon hierarchy for widgets within one application
electric organ timbre
Window events
pitch C_2, higher intensity
Other widgets (see above)
Activate window
180 msec duration

Figure 11. The hierarchy of earcons used in the sonically-enhanced windows.

Audio feedback was also used to overcome the problem of discontinuity of input. The event of switching from one window to another was marked by a short, high-pitched note in the new window's timbre at increased intensity. For example, when switching from another application to our example one, an electric organ note at pitch C_2 (523Hz) was played for 180 msec. This indicated to the user if, for example, the mouse had been knocked into another window by mistake. Other window events, such as dragging or re-sizing the window were also indicated in the same way.

Sonically-enhanced 'drag and drop'

'Drag and drop', although not strictly a widget, is a feature common to most interfaces. It allows the user to select an object in the interface, drag it and then drop it over something else. For example, the user might select a text file icon and then drop it on the icon for a text editor to edit it. In many systems the target highlights when the mouse is over it to indicate that an object may be dropped. This is a very natural way of interacting and provides the user with great flexibility; he/she can drag the text file to different applications, disks, printers or the wastebasket as required. There are several problems with this. It can be difficult to hit a target and drop things on it (especially if the target is small). Gaver (1989) noted this problem in the Macintosh Finder and attempted to solve it in his *SonicFinder* - a version of the Finder which used non-speech sounds called *Auditory Icons* (p 81): "Auditory confirmation that a target has been hit turns out to be one of the most obviously useful features of the SonicFinder, especially in finding small folder icons that may be partially obscured by overlapping windows. A common problem in hitting such targets comes when the object, but not the cursor, is positioned over the target. In this situation, dropping the object does not place it inside the target, but instead positions it so that it obscures the target further. The auditory icon indicates a true hit, and so reduces the amount of time spent playing 'chase the trashcan'".

It is also difficult to know if the object being dragged can be dropped on a particular application. For example, a graphics file cannot be dropped on a text editor. The application icon may not highlight when the mouse is over it to indicate that it cannot open but this lack of feedback can be confusing and is exactly the same as when the mouse is not over the target, creating more confusion. In some systems the target will highlight but nothing will happen if the object is of a type the target cannot deal with, again this is confusing.

Sounds can overcome this problem. Gaver (1989) used a clinking sound in his SonicFinder to indicate when an item was over a target application. A similar approach was used here. When an item was dragged over a target that would accept it a quiet background sound was played (similar to the earcon for when the mouse was over a sonically-enhanced button). This indicated to users that the item could be dropped. This also cued them as to when to drop the item. The sound indicated they were over a legal target, no sound indicated that they were not over the target, or the target would not accept the item. The timbre of the earcon played was that of the application the mouse was over. The sound was low-intensity (to avoid annoyance) and was played for as long as the mouse was over the target application icon.

Results of sonic-enhancement

Many of these widgets (or earlier versions of them) have been experimentally tested to discover if the sonic enhancements improved usability. The results have been very promising (Brewster, 1994). For example, the sonically-enhanced buttons were given a significantly higher overall preference rating by users. The time and number of mouse-clicks needed to recover from slip-off errors were both significantly reduced. This was not at the expense of making the buttons more annoying to use as there was no difference in terms of annoyance between standard graphical buttons and the sonically-enhanced ones. These results indicated that, if sonically-enhanced buttons were included in an interface toolkit, slip-off problems could be dealt with very effectively (Brewster *et al.*, 1995a).

As another example, sonically-enhanced scrollbars were tested and found to significantly reduce time taken to complete certain tasks. As with sonically-enhanced buttons, error recovery times were significantly reduced. Sonically-enhanced scrollbars were again given a significantly higher overall preference rating by users. They also significantly reduced the mental workload felt by users when performing scrolling tasks (Brewster *et al.*, 1994). There was no increased annoyance due to the sounds. This again indicates that the integration of auditory feedback into graphical widgets is likely to provide more usable interfaces.

Implementation of the widgets and future work

The earcons used in the sonically-enhanced widgets were controlled by MIDI (Roads, 1996). Almost all modern computer systems either have built-in MIDI-controlled synthesisers (for example on sound cards or via DSP chips) or can easily be connected to them. The widgets described below have been run both on external synthesisers and the internal software synthesiser which is part of the Macintosh operating system. The widgets all use the General MIDI standard (Roads, 1996) so that they can be played on any synthesiser that supports the standard. The widgets send data in real-time to the synthesiser. MIDI data is compact, with usually only three bytes needed to control a sound, so the computational overhead is low. This means that even slow machines (for example, Network Computers) could use the widgets with an external synthesiser attached. Using MIDI also provides a way for users to customise the sounds. Standard synthesiser control software can be used to change the timbre or intensity of the sounds in any widget.

The sonic-enhancements were implemented on top of standard Macintosh graphical widgets. The widgets had code added to them to play the earcons as interactions are taking place. The current versions of the widgets have the MIDI data to play the sounds hard-coded into them. This is inflexible but allowed them to be tested to assess the effectiveness of the sonic-enhancements. Future versions will have the MIDI data stored as resources (in the same way as text and graphics in graphical interfaces). This will allow the sounds to be changed by simply editing a resource file. In the widget's program code there will be calls to the resource file to read in the MIDI data, this will then be used to generate the sounds necessary.

Once a complete set of sonically-enhanced widgets has been built and any problems with them ironed-out, a set of *auditory themes* will be developed. Graphical themes are common to many systems (for example, Microsoft Windows95). With graphical themes the user does not set the colour of each interface element but chooses a theme for the whole interface. This saves the user time and also allows for harmonious colours to be chosen that do not clash. Auditory themes would control the sounds used in a similar way. Different themes would be provided (for example there might be a rock theme or a blues theme) in which all of the sounds for the different widgets would be designed as a coherent set by the interface designer. This would avoid users choosing sounds that clashed or sounded bad. Storing the MIDI data in resource files will simplify the use of themes. The new theme data can be placed into the resource file to overwrite the previous theme.

Another important step is to combine the widgets into a complete toolkit. They have been designed with this in mind so that this should be simple and there should be few sonic conflicts (such as where one widget requires silence but another needs a sound to be played). For example, in the sonically-enhanced windows above, one group of auditory widgets can provide some of the feedback needed by another. Once the toolkit has been built, applications with sonically-enhanced interfaces will be much easier to create. The sonically-enhanced widgets will be used in the same way as graphical widgets, i.e. called from within the source code of the application program. This will not be a high-level way of creating interfaces like an User-Interface Design Environment (UIDE) (Myers *et al*., 1990), which often allow graphical specification of layout or constraint based programming, but it will be an important first step in enabling the simple construction of sonically-enhanced interfaces.

In the X Windows system *geometry managers* are used to control the layout of graphical widgets and avoid graphical conflicts such as one widget being drawn on top of another (Ousterhout, 1994). These managers provide simple control of layout, for example ensuring the size of a button is large enough to display its label, or laying items out horizontally or vertically as specified in the layout policy (Ousterhout, 1994). A similar approach can be taken in sound: *Sound managers* will be created that will manage the sounds that must be played. Auditory 'layout policies' can be constructed so that sounds can be assigned to actions and any sonic conflicts that occur can be dealt with. This might be something

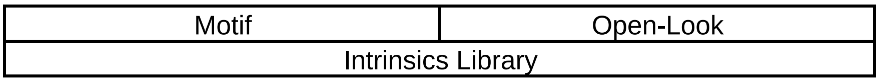

Figure 12. Layered structure of X Windows toolkits.

simple like assigning the pitch of a note to the position of the scrollbar thumbwheel. The manager will control the maximum and minimum values of the scrollbar and so can set the maximum and minimum ranges of the notes needed to represent it.

X Windows toolkits use a layered structure with a library of commonly used functions, *intrinsics*, and then a toolkit with a particular look-and-feel built upon this (see Figure 12). This simplifies the implementation of toolkits as many basic functions are in the intrinsics library. When the toolkit of widgets has been constructed it will be clear what the basic functions are that are likely to be needed in other toolkits. These will be then added to an audio intrinsics library. This will then facilitate the creation of other sonically-enhanced widget sets because the basic functionality necessary will be in place.

Conclusions

Currently it is very difficult to add sounds to human-computer interfaces to improve usability. One reason for this is that the area of sound in human-computer interaction is in its infancy and there are few examples or guidelines on which designers can build. Another problem is that most designers are not sound experts so do not know how to use sound effectively. The work described in this paper has shown, in detail, how non-speech sounds called earcons can be added to a set of standard graphical widgets to improve usability.

Each of the widgets in a standard widget set was analysed to find usability problems. Many usability problems occur because users cannot look at two places at once. Users really want to look at the information involved in their current task rather than at interface components. Using sound allowed them to concentrate their visual attention on the information they were interested in and get other feedback through sound, it did not force them to take their eyes off their task and did not allow the interface to intrude into the task being performed. Earcons were added to buttons, scrollbars, menus, alert boxes, windows and drag and drop. The structure of the earcons used was well defined and based on the structure of widgets in a standard graphical interface toolkit, using many of the advantages that this brought (for example feedback from one widget lower down in the hierarchy helping to solve a usability problem in a widget higher up). Designers can use these sonically-enhanced widgets described here in their interfaces knowing that sound will be used effectively and coherently, and usability will be improved.

Acknowledgements

This work was partially supported by EPSRC Grant GR/L79212. Many thanks to the referees and their useful comments.

References

Arons, B. and Mynatt, E. (1994). The future of speech and audio in the interface. *SIGCHI Bulletin*, **26**, 4, 44-48.

Barfield, W., Rosenberg, C. and Levasseur, G. (1991). The use of icons, earcons and commands in the design of an online hierarchical menu. *IEEE Transactions on Professional Communication*, **34**, 2, 101-108.

Beaudouin-Lafon, M. and Conversy, S. (1996). 'Auditory illusions for audio feedback' in **Tauber, M. (ed)**, *ACM CHI'96 Conference Companion* ACM Press, Addison-Wesley, 299-300.

Berglund, B., Preis, A. and Rankin, K. (1990). Relationship between loudness and annoyance for ten community sounds. *Environment International*, **16**, 523-531.

Blattner, M., Sumikawa, D. and Greenberg, R. (1989). Earcons and icons: Their structure and common design principles. *Human Computer Interaction*, **4**, 1, 11-44.

Brewster, S.A. (1994) *Providing a structured method for integrating non-speech audio into human-computer interfaces*. PhD Thesis, University of York, UK.

Brewster, S.A. (1997). Using Non-Speech Sound to Overcome Information Overload. *Displays*, **17**, 179-189.

Brewster, S.A. and Crease, M.G. (1997). 'Making Menus Musical' in **Howard, S., Hammond, J. and Lindgaard, G. (eds)**, *Proceedings of IFIP Interact'97* Chapman & Hall, 389-396.

Brewster, S.A., Raty, V.-P. and Kortekangas, A. (1996). 'Earcons as a method of providing navigational cues in a menu hierarchy' in **Sasse, A., Cunnigham, R. and Winder, R. (eds)**, *Proceedings of BCS HCI'96* Springer, 169-183.

Brewster, S.A., Wright, P.C., Dix, A.J. and Edwards, A.D.N. (1995a). 'The sonic enhancement of graphical buttons' in **Nordby, K., Helmersen, P., Gilmore, D. and Arnesen, S. (eds)**, *Proceedings of IFIP Interact'95* Chapman & Hall, 43-48.

Brewster, S.A., Wright, P.C. and Edwards, A.D.N. (1992). 'A detailed investigation into the effectiveness of earcons' in **Kramer, G. (ed)**, *Proceedings of ICAD'92* Addison-Wesley, 471-498.

Brewster, S.A., Wright, P.C. and Edwards, A.D.N. (1993). 'An evaluation of earcons for use in auditory human-computer interfaces' in **Ashlund, S., Mullet, K., Henderson, A., Hollnagel, E. and White, T. (eds)**, *Proceedings of ACM/IFIP INTERCHI'93* ACM Press, Addison-Wesley, 222-227.

Brewster, S.A., Wright, P.C. and Edwards, A.D.N. (1994). 'The design and evaluation of an auditory-enhanced scrollbar' in **Adelson, B., Dumais, S. and Olson, J. (eds)**, *Proceedings of ACM CHI'94* ACM Press, Addison-Wesley, 173-179.

Brewster, S.A., Wright, P.C. and Edwards, A.D.N. (1995b). 'Experimentally derived guidelines for the creation of earcons' in **Kirby, M., Dix, A. and Finlay, J. (eds)**, *Adjunct Proceedings of BCS HCI'95* , 155-159.

Brown, M.L., Newsome, S.L. and Glinert, E.P. (1989). 'An experiment into the use of auditory cues to reduce visual workload' in *Proceedings of ACM CHI'89* ACM Press, Addison-Wesley, 339-346.

Dix, A., Finlay, J., Abowd, G. and Beale, R. (1993). *Human-Computer Interaction*. Prentice-Hall.

Dix, A.J. and Brewster, S.A. (1994). 'Causing trouble with buttons' in *Ancillary Proceedings of BCS HCI'94* Cambridge University Press.

Edworthy, J., Loxley, S., Geelhoed, E. and Dennis, I. (1989). The perceived urgency of auditory warnings. *Proceedings of the Institute of Acoustics*, **11**, 5, 73-80.

Gaver, W. (1989). The SonicFinder: An interface that uses auditory icons. *Human Computer Interaction*, **4**, 1, 67-94.

Gaver, W., Smith, R. and O'Shea, T. (1991). 'Effective sounds in complex systems: The ARKola simulation' in **Robertson, S., Olson, G. and Olson, J. (eds)**, *Proceedings of ACM CHI'91* ACM Press, Addison-Wesley, 85-90.

Harrison, M. and Barnard, P. (1993). 'On defining requirements for interaction', *Proceedings of the IEEE International Workshop on requirements engineering*. IEEE, 50-54.

Karshmer, A., Brawner, P. and Reiswig, G. (1994). 'An experimental sound-based hierarchical menu navigation system for visually handicapped use of graphical user interfaces' in *Proceedings of ACM ASSETS'94* ACM Press, 123-128.

Lee, W.O. (1992). 'The effects of skill development and feedback on action slips' in **Monk, A., Diaper, D. and Harrison, M.D. (eds)**, *Proceedings of HCI'92* Cambridge University Press, 73-86.

Mansur, D.L., Blattner, M. and Joy, K. (1985). Sound-Graphs: A numerical data analysis method for the blind. *Journal of Medical Systems*, **9**, 163-174.

Moore, B.C.J. (1997). *An Introduction to the Psychology of Hearing*. (4th ed.) Academic Press.

Myers, B. (1990). All the widgets. *ACM SIGRAPH Video Review*, **CHI'90 Special Issue**, 57,

Myers, B.A. (1991). 'State of the art in user interface software tools', in **Hartson, H.R. and Hix, D., (eds)**, *Advances in Human-Computer Interaction* Ablex Publishing.

Myers, B.A., Guise, D., Dannenberg, R., Vander Zanden, B., Kosbie, D., Pervin, E. and Mickish, A. (1990). Garnet: Comprehensive support for graphical, highly-interactive user interfaces. *IEEE Computer*, 71-85.

Mynatt, E.D. (1992). 'Auditory presentation of graphical user interfaces' in **Kramer, G. (ed)**, *Proceedings of ICAD'92* Addison-Wesley, 533-555.

Mynatt, E.D. and Weber, G. (1994). 'Nonvisual Presentation of Graphical User Interfaces: Contrasting Two Approaches' in *Proceedings of ACM CHI'94* ACM Press, Addison-Wesley, 211.

Ousterhout, J.K. (1994). *Tcl and the Tk toolkit*. Addison-Wesley.

Patterson, R.D. (1982). *Guidelines for auditory warning systems on civil aircraft* (CAA Paper No. 82017). Civil Aviation Authority, London.

Perrott, D., Sadralobadi, T., Saberi, K. and Strybel, T. (1991). Aurally aided visual search in the central visual field: Effects of visual load and visual enhancement of the target. *Human Factors*, **33**, 4, 389-400.

Pitt, I.J. and Edwards, A.D.N. (1991). 'Navigating the interface by sound for blind users' in **Diaper, D. and Hammond, N. (eds)**, *Proceedings of BCS HCI'91* Cambridge University Press, 373-383.

Pitt, I.J. and Edwards, A.D.N. (1995). 'Pointing in an auditory interface for blind users' in *Intelligent Systems for the 21st Century: Proceedings of the 1995 IEEE International Conference on Systems, Man and Cybernetics* IEEE, 280-285.

Portigal, S. (1994) *Auralization of document structure*. MSc. Thesis, The University of Guelph, Canada.

Reason, J. (1990). *Human Error* Cambridge University Press.

Reichman, R. (1986). 'Chapter 14: Communications paradigms for a window system', in **Norman, D.A. and Draper, S.W. (eds)**, *User-Centered System Design*. Lawrence Erlbaum Associates, 285-314.

Roads, C. (1996). *The computer music tutorial* MIT Press.

Savidis, A. and Stephanidis, C. (1995). 'Developing dual user interfaces for integrating blind and sighted users: The HOMER UIMS' in **Katz, I., Mack, R. and Marks, L. (eds)**, *Proceedings of ACM CHI'95* ACM Press, Addison-Wesley, 106-113.

Sumikawa, D., Blattner, M., Joy, K. and Greenberg, R. (1986). *Guidelines for the syntactic design of audio cues in computer interfaces* (Technical Report No. UCRL 92925). Lawrence Livermore National Laboratory.

Wenzel, E.M. (1992). 'Spatial sound and sonification' in **Kramer, G. (ed)**, *Proceedings of ICAD'92* Addison-Wesley, 127-150.