December 17, 2006

Research Internship

Navigational Aid for the Visually Impaired

DA-IICT

Prof R.N Biswas

Deepak Jagdish, Mohit Gupta, Rahul Sawhney, Shreyas Nangia

Summer Research Internship Report

Abstract:

Contents

Introduction

The modern world has been designed with a certain view of the human who is to benefit from the construction. In this process of engineering our world, we have at certain junctures not thought about minorities such as those who are not physically or mentally able to conform to our supposed regularities. Our project aimed to create and describe experiments in auditory information processing, in addition to dealing with the technological and design issues involved in developing navigational solutions for the visually impaired.

We looked at the problem in two broad areas where navigation for the blind is limited due to designs that have been streamlined for ocular use – *traveling in modern day interiors and urban spaces* and *the use of spatially aware systems on the computer*.

The project aimed to provide a better understanding of the technical and social issues which hinder in easy navigation for the visually impaired. The endeavor was to gain an insight into the understandings and perceptions of the visually impaired and to continually prototype and explore our ideas for technical feasibility. An initial expectation was also to allow for free brainstorming continually during the duration of the project about techniques and interfaces that could be built, without a firm focus on products already being developed. It was felt that features and little nuances of interaction should be explored without allowing for too many constraints and preconceived notions about how things should be done.

With the goal of solving the problem statement illustrated above, the following **tasks that were accomplished** and the issues and processes leading to the same have been discussed in this report.

- A basic proof-of-concept was developed which included a *sound-grid based mouse feedback interface*, *ultrasonic sensors controlled through digital pulses from a microcontroller* and a *familiarity with interfaces of Windows Mobile*. Animated previews of concepts such as SonicMap and in-Focus menu were also created to explain the concepts and obtain user feedback for the same.

- Research papers and studies by various groups were meditated upon to gain an understanding of the problem areas, complexities involved and limitations for development of the prototype and its use by the visually impaired. A multi-disciplinary study was essential for the filling in the details on the identity and perceptions of the blind; these studies required familiarity in the areas of Human

Computer Interaction, Sense modalities, Techniques of User Research and Usability Engineering. Students from NID and the work of Abhigyan Singh and Rahul Mukherjee on Synesthesia helped us through the processes of interacting with the blind for user research and usability.

- DirectX worlds were created to help us model the auditory feedback before the actual hardware device was completed for testing. The interaction and auditory responses of our system were streamlined based on user feedback and further features were added to present a more complete representation of the operating system augmented with accessibility features. This involved a solid understanding of the workings of the Windows operating system and its structures for issuing interrupts and events at different junctures in various modes of operation.
- The hardware was refined for better response and noise reduction. The hardware circuit went through multiple iterations for it to provide stable readings and to calibrate the signal amplifications and timings. Certain issues regarding the unhandled hardware interrupts in the microcontroller were addressed and the PromiESD Bluetooth module was used to transfer the data from the microcontroller to a Windows Mobile based device (Windows based PC used for experimentation purposes). The hardware was prepared for PCB layout and the proper components were soldered on to the PCB as a further step towards a portable system. Battery issues were not looked upon with care, and therefore changes were required to accommodate the fluctuating voltage and current response of standard batteries.
- Familiarization with new platforms such as the Windows Presentation Foundation by active involvement in the beta-testing of development environments like Expression Interface Designer to create in-Focus menu. These platforms were subsequently leveraged to showcase possibilities of in-Focus menu in the future.
- Interfaces for the application running on mobile device (loaded with Windows Mobile 5.0 OS) were designed in Visual Studio 2005 keeping in mind the usability aspects for the visually impaired.
- A physical model made of acrylic was also designed with the help of product designers from NID, to test the morphology of the device and its accompanying circuit in real environments.

Process Description

The design and development of the current prototype has progressed through three distinct stages:

- Ideation, user expectations and system level framework;
- Technical explorations and user feedback;
- Research relating to the visually impaired, development of usable software systems and refinements towards a hardware prototype.

The initial brainstorming sessions were aimed to broaden the teams understanding of problems faced by the visually impaired and the details of such experiences. Enhancements rather than direct solutions were considered as a more optimal means of aiding the visually impaired in navigation. These brainstorming sessions were interleaved with visits to the local blind school in Gandhinagar to help us draw up a structure of the expectations from our designs. The need for understanding auditory processing by the blind led us to consider computer-based simulations of spaces; such investigations helped us to discuss the problems faced by the visually impaired in using modern GUI based operating systems and software applications. The goals were broadened to include mouse based navigation and the entire system framework was prepared with user scenarios, design patterns and considerations for further research.

The second phase of development involved exploring technologies which could help realize the system goals. The basic set of expectations that resulted from the documentation developed in the first phase was used a starting point to consider various hardware and software components that were available. Sensors, microcontroller and development environments and work-division were decided upon and development was started on multiple-module basis. The work primarily involved learning and familiarizing with the tools and techniques that were to be useful in the development of the prototype. A basic proof of concept of the hardware and software was presented in Bangalore.

This document reports our explorations, thinking and findings in the third phase. The work started as a critical look at the methods of interaction we were using and attempts to better understand the hardware functionality and methods of increasing accuracy and stability of the system. The results at the end of this phase have been summarised in the introduction as a broad look into the accomplishments and findings of the project.

Team and Contributions

Deepak Jagdish

Design and development of interfaces for the software application using .NET Framework 3.0 (containing Windows Presentation Foundation)
Exploring systems design of existing solutions for the visually impaired;
Research for base technologies to be used in different modules of the project which included Expression Suite, Microsoft Orcas, Visual Studio 2005 etc.;
Development of Bluetooth module for PC & PocketPC for testing and experimentation purposes of the PromiESD Bluetooth chip; also studied alternative methods to implement the same.

Mohit Gupta

User research, usability feedback and HCI-related research
Interaction of the microcontroller and Bluetooth module
Mobile Interfaces (not used) and visualizers for easy demonstration
Soldering and testing of the PCB module and design of the physical prototype

Rahul Sawhney

Software Design which involved:
Making windows accessible to the blind
Design and development of DirectX based virtual worlds
Research and use of technologies (MS Active Accessibility , Win32 , .NET , DirectX SDK, WMP API , Active X - COM , Windows Event handling , Process handling and inter process communication)

Shreyas Nangia

Experimentations with the circuit for better working of the ultrasonic sensors and reduction of signal noise
Development of embedded software for the ultrasonic range finder
Design and production of the PCB
Visual radar for real-time data collected through sensors

Motivations

The problems faced by the blind/visually impaired are many, ranging from simple tasks like getting dressed or making a cup of tea, to problems like inability to read and write easily. But one of the biggest challenges caused by blindness is Isolation, both physical and social. The problem stems from the fundamental challenge of self-controlled mobility. It is extremely difficult for the blind/visually impaired to even get out of their own homes and navigate themselves to places they wish to go. In addition to such constraints in the physical world, the blind people also have near-zero access to the plethora of facilities provided by the advent of technology. In fact, upon interaction with some blind people, it was clear that they were not able to use some of the most common facilities like instant messaging over phone, accessing the internet, using media players etc. Their access domain to today's common technologies is nearly non-existent and this is one of the primary issues we propose to solve effectively.

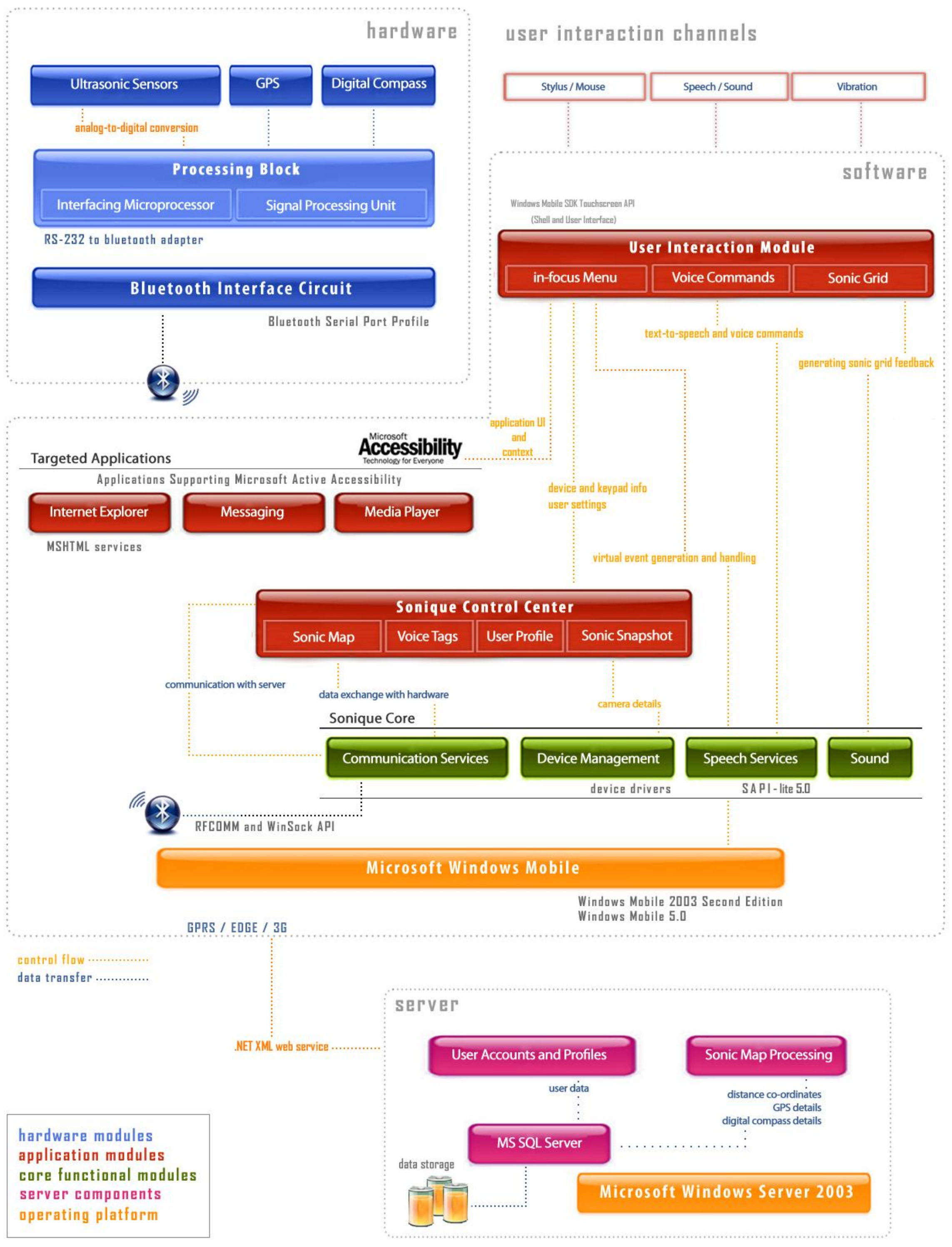
In this endeavour of ours, we aim to see health as expanding possibilities for the blind/visually impaired struggling to lead a normal, comfortable life. We feel the need to enable the blind/visually impaired people to achieve a high quality of access to day-to-day navigation scenarios and techniques. This project effort proposes to give the blind the ability to move around easily in the real-world and in the process revolutionize their sense of mobility. In addition to this, it also aims to give the blind people their personal space in the virtual world. With innovations in virtual-world applications and navigation techniques, Sonique makes anything in the virtual domain available to the blind user by providing easy accessibility solutions on the PocketPC. The impact of our solution is envisioned to be such that the blind community can break the barriers of their restricted environment and attain total freedom of navigation, both in the real and virtual worlds.

Designing with the above vision, the gave us the chance to come up with some really new and exciting ways to use existing technologies, combine them and optimize it for use by blind people.

Expectations and overview after the first phase

An overview of the innovative solutions that we have pictured is described in this section. Each of these modules, when integrated with other components of the Sonique solution gives a whole new dimension of vision to the blind people. The major modules are shown in perspective with respect to other elements in the System Architecture diagram.

. Sonique . System . Architecture .



As per our initial project plans, the design of solution to be constructed was as shown in the System Architecure diagram. Out of this initial blueprint, only a select number of items have been able to be implemented, either due to resource constraints or time constraints.

In the hardware section, it was initially decided to include Ultrasonic Sensors, a GPS tracker, and a Digital Compass. In the final product, only the ultrasonic sensors were included which provided basic system functionalities. Items like GPS tracker and Digital Compass required further time for implementation and hence were only discussed at a technical level without implementation. We identified that for the final real-world navigation system to work successfully, it was necessary to have a *reference point* on which the system could map positional coordinates. This is why a GPS device and a Digital Compass were deemed to be necessary.

The next task that was planned was to interface this set of ultrasonic sensors with a microcontroller which would make sense of the data collected by the sensors. This was successfully implemented with an Atmel ATMEGA32 microcontroller working simultaneously with three pairs of ultrasonic sensors (transmitter-receiver pair). The received data was verified to be correct in its scope of around 3.5 meters, while the resolution of the sensors was very hard to estimate.

Our next aim was to transfer this distance data to a mobile device like a PDA (or Smartphone) wirelessly. For this, we preferred using the Bluetooth® wireless protocol. The PDA would have inbuilt Bluetooth capability, which meant that the microcontroller would have to interface itself with a Bluetooth chip for pairing up with the PDA. The PromiESD02 Bluetooth chip was chosen because it had an inbuilt Bluetooth stack which would handle incoming connections, and it also proved to be much cheaper than other industrial alternatives. Since it was the first time that we were working with such a chip, there were some mistakes made in implementing it and so this stage took much longer than expected. Finally, our aim was achieved, of transferring data wirelessly to a Bluetooth enabled device running an operating system which would host our client software.

Moving onto the software domain, we had to build software for a mobile device like the PDA running on a Windows Mobile operating system. Our estimated time for completion of this stage was met fairly accurately, with the removal of certain initially planned modules due to lack of certain software libraries. Even then, we were able to implement software that could receive data on-the-fly over Bluetooth and this data could then be used for direct sound modulation. From this point onwards, our plans had to change from the initial blueprint because the rest of the system had to be implemented on a laptop rather than a mobile device like a PDA. This was to enable a better demonstration of the prototype as well as financial restraints of buying software libraries for a mobile device.

Software design and development

With Sonique, we intended to redefine the way the visually impaired used their computers, it was necessary for us to change the input and output mechanisms fundamentally, without introducing new peripherals or redesigning the existing ones.

Functionally, we revamped the following Windows functionalities:

- *Mouse functionality*
- *Keyboard functionality*
- *Sound Feedback system*
- *Windows Event Management System*
- *Windows Accessibility at OS and Application levels*

Mouse

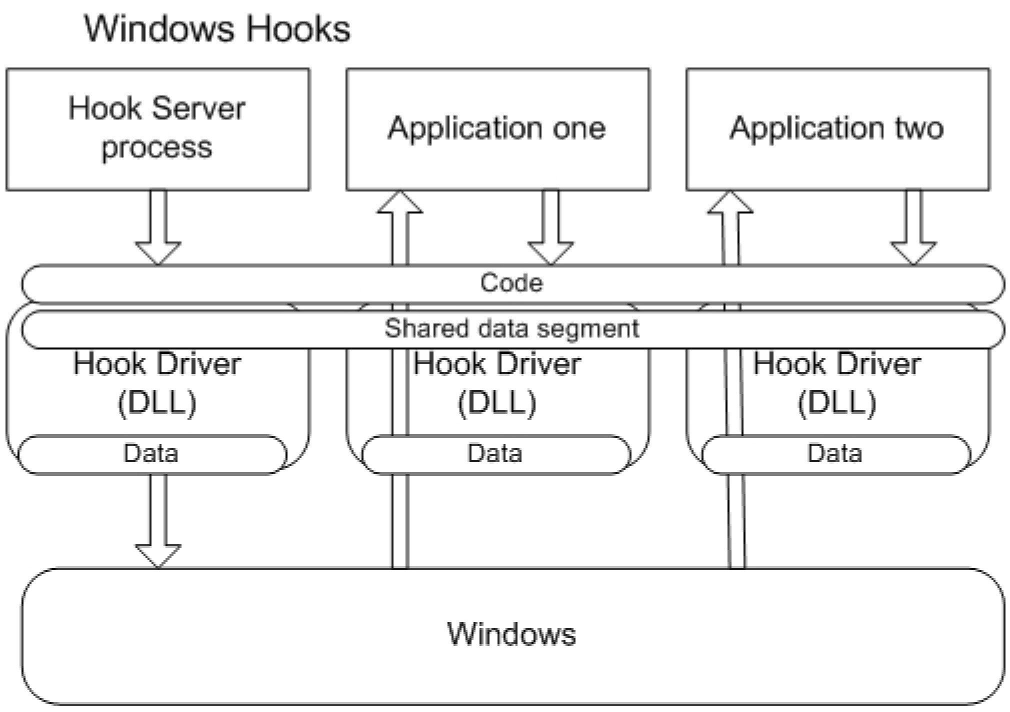
The visually impaired have never been able to use a mouse or any such interaction device which provided non-linear or spatial interaction. Thus , it was required that whatever inputs the mouse provided were first intercepted by the software layer of Sonique, processed, interpreted, actions performed and then , if required sent to the Operating System (OS) Windows.

To accomplish the above task, we made use of Hooks that are present in the Operating System. In the following paragraphs, our approach to learning and implementing hooks to address this issue is illustrated:

In the Microsoft® Windows® operating system, a hook is a mechanism by which a function can intercept events (messages, mouse actions, keystrokes) before they reach an application. The function can act on events and, in some cases, modify or discard them. Hooks modify the actual flow of code. A hook is ultimately a callback function that applications register with a particular system event. Functions that receive events are called filter functions and are classified according to the type of event they intercept. For example, a filter function might want to receive all keyboard or mouse events. For Windows to call a filter function; the filter function must be installed—that is, attached—to a Windows hook (for example, to a keyboard hook). Attaching one or more filter functions to a hook is known as setting a hook. If a hook has more than one filter function attached, Windows maintains a chain of filter functions. The most recently installed function is at the beginning of the chain, and the least recently installed function is at the end.

A fundamental aspect of hooks that we utilized is their scope. Normally, hooks may have either system or thread scope. A few, however, can only have system scope. When a hook works at the thread level, it can only trap events generated within that thread. For example, a keyboard hook gets invoked only for the keystrokes directed to the thread's input queue. Similarly, a systemwide mouse hook gets called whenever the user moves the mouse, regardless of the particular thread that handles the event. A system-scoped hook is called to handle the event for all the currently running threads. This poses a precise context problem. How can a piece of code defined in one Win32 process invade the memory space of another process? To allow for this, a systemwide hook must be defined in a DLL so that the system can easily inject that code into each of the Win32 process memory spaces.

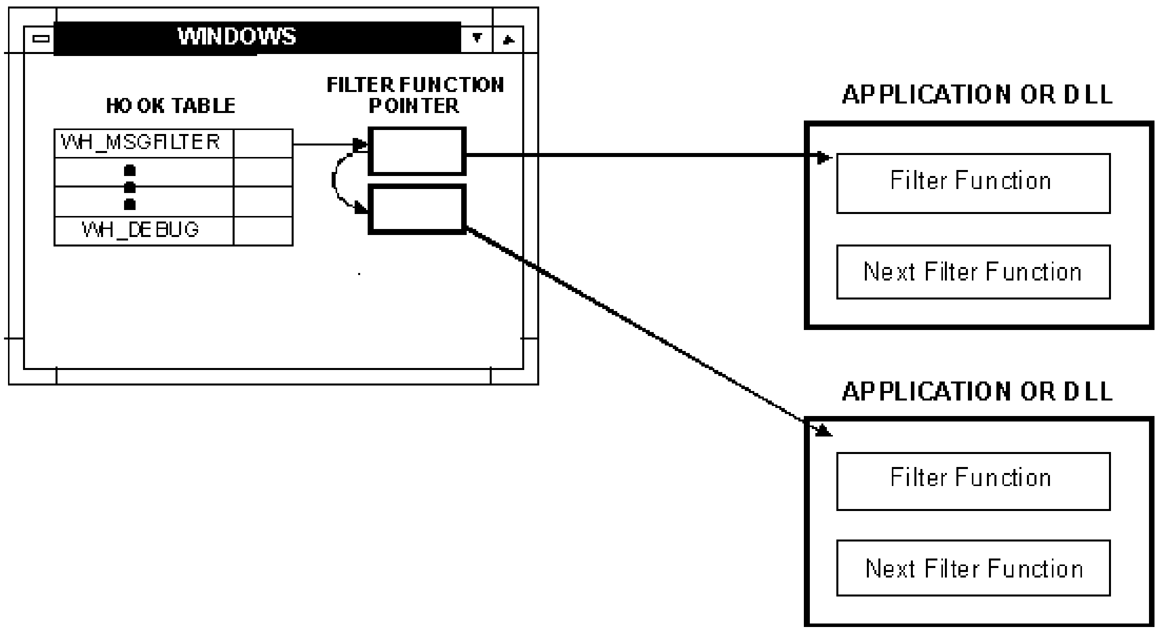
Thus thread hooks (or local hooks) are patently more efficient than system hooks (or global hooks). On the other hand, they cannot accomplish all the tasks global hooks can.



The .NET Framework does not provide built-in facilities or infrastructure to handle hooks. Right now, hooks are considered merely a special breed of callback functions and their implementation is left to P/Invoke (the .NET Framework infrastructure to call unmanaged APIs residing in the underlying operating system).

There is, as we realized an important platform-related aspect that marks Win32 and the .NET Framework. In Win32, the smallest unit of processing is the process. The CPU works by allotting slices of time to each process. While running, a process sees the whole 32-bit range of memory at its disposal. For this reason, it's architecturally impossible for a process to inadvertently corrupt another process' memory. (A process can still break another process, but they must both be explicitly using globally shared memory.)

In the .NET Framework, there is a significant change to this process. The .NET managed code runs under the control of the common language runtime (CLR) module and is subject to inspection and verification before execution. The CLR enables a piece of managed code only if it can be marked as type-safe code. The verification process ascertains the correctness of the intermediate language (IL) code and ensures that it accesses only authorized memory locations. In addition, type-safe code is guaranteed to reference only strictly compatible types and call objects only through properly defined types.



Thus our first problems were on how to get a mouse hook (or keyboard) run in .NET environment. We found several code snippets on the internet which got us started. But the .NET environment was completely new to us (it is a relatively new application development platform) and it took us a while to attain a certain comfort level with it. Then we spent a lot of time understanding Windows hooking nuances. Hooks, as we realized was a technique which interfered with the normal working of application level threads and in our case we had to monitor all the inputs being sent to the OS in general.

The normal, high-level keyboard hook, WH_KEYBOARD, intercepted keystrokes as they were removed from a thread's message queue. The WH_KEYBOARD hook works well for most applications. However, certain keystrokes are never directed to a thread's message queue. The Ctrl+Esc, Alt+Tab, and Alt+Esc key combinations are perfect examples. These keystrokes are handled internally by the system's raw input thread. Since application threads never receive messages for these keystrokes, there is no way that an application can intercept them and prevent the normal processing. This behavior is by design and ensures that a user can always switch to another application's window even if an application's thread enters an infinite loop or hangs.

However, there is a small class of applications that really has a valid need to intercept these keystrokes. To meet the needs of these applications, Microsoft introduced the WH_KEYBOARD_LL hook. This low-level hook is notified of keystrokes just after the user enters them and before the system gets a chance to process them. But this hook has a serious drawback: the thread processing the hook filter function could enter an infinite loop or hang. If this happens, then the system will no longer process keystrokes properly and the user will become incredibly frustrated.

To alleviate this situation, Microsoft places a time limit on low-level hooks. When the system sends a notification to a low-level keyboard hook's filter function, the system gives the function a fixed amount of time to execute. If the function does not return in the allotted time, the system ignores the hook filter function and processes the keystroke normally. The amount of time that is allowed (in milliseconds) is set via the LowLevelHooksTimeout value under the HKEY_CURRENT_USER\Control Panel\Desktop registry subkey.

In our case though, our hook (low level) was intercepting *all* messages being sent to the OS, hence it was imperative for us to handle the messages carefully and have a bypass mechanism in place, in case Sonique software module fell into a no-exit subroutine.

Here, ***nCode*** provides us with the escape clause:

```
if (nCode >= 0 )

            {

                try

                {

                    mouseHookStruc =
(MouseHookStruct)Marshal.PtrToStructure(lParam,
typeof(MouseHookStruct));


                }


                catch (Exception e)

                {

                    MainForm.handle.LogWrite(Convert.ToString(e));

                    return 1;

                }


                // MainForm.handle.LogWrite("doneproc");

                // if ok and someone listens to our events
```

Basically, if ***nCode*** is less than zero (which would be the case when Ctrl + Alt + Del) the keys ctrl, alt, del are allowed to reach the OS.

Then the information we needed about mouse inputs needed to be extensive , about all it's buttons including wheel and scrolling. That required us to define a structure which would be able to accomodate everything the hook callback provided. The following structure was hence utilized.

```
public class MouseHookStruct

        {

            public Point pt;

            public int mouseData;

            public int flags;

            public int time;

            public int dwExtraInfo;

        }
```

```
mouseHookStruc = (MouseHookStruct)Marshal.PtrToStructure(lParam,
typeof(MouseHookStruct));
```

The above line basically “Marshalls” the information from a C style struct (which is the way in which windows (Win 32) stores information) to our own structure MouseHookStruct.

What we then did was to , depending on the mouse inputs , perform different actions. The following are the starting lines of one of our sub routines.

```
   # region menu



                if (kickmup && kickrup)

                {

                    if (wParam == WM_MBUTTONUP) { kickmup = false;
return 1; }

                    if (wParam == WM_RBUTTONUP) { kickrup = false;
return 1; }

                }
                try
                {

                    if (freeze)
                    {
                        if (!selected)
                        {
                            switch (wParam)
                            {

                                case WM_MBUTTONUP: if (j == 0) {
TTS.speak("Settings Menu , use scroll wheel"); return 1; }
                                    selected = true;
mouse_event((uint)MOUSEEVENTF.MIDDLEUP, 0, 0, 0, 0); return 1; break;


                                case WM_RBUTTONUP: if (j == 0) {
TTS.speak("Settings Menu , use scroll wheel"); return 1; }
                                    selected = true;
mouse_event((uint)MOUSEEVENTF.MIDDLEUP, 0, 0, 0, 0); return 1; break;


                                case WM_LBUTTONUP: if (j == 0) {
TTS.speak("Settings Menu , use scroll wheel"); return 1; }
                                    selected = true;
mouse_event((uint)MOUSEEVENTF.MIDDLEUP, 0, 0, 0, 0); return 1; break;


                                case WM_MOUSEWHEEL:
                                    if (mouseHookStruc.mouseData < 0)
if (++j > 4) j = 1;
                                    if (mouseHookStruc.mouseData > 0)
if (--j < 1) j = 4;
```

Care was taken not to hinder the with the regular functionality of the buttons. More functionality was added by utilizing a sequence of clicks to perform various actions. Windows basically utilizes one mouse button at a time. We imparted functionality to several combinations of mouse clicks; a middle button click after a right click would work differently than a right click after middle click.

```
case WM_RBUTTONDOWN: if (isL && !infocus)
                            {
                              if (!isLM && !rdown && !mdown)
                                {


                               MainForm.handle.LogWrite(" R  over L ");
```

Basically, our aim was to make the desktop more accessible through the mouse. The various functions provided aimed to provide the user with a lot of accessibility information depending on the mouse cursor position and functional modes.

The main challenge here lay in how to handle the intercepted input. We also had to process fast as another input was always on it's way to the callback function. Thus it required us design a really complicated logical anaylser which would quickly process inputs or rather combination of inputs and start of relevant actions before the next input comes in.

Keyboard

Same as what we did with the mouse, low level Keyboard hooks were deployed to fully intercept all keyboard inputs. Care was taken to store the keyboard state in the beginning, because Windows allows for sticky keys, key toggling etc.

```
byte[] keyState = new byte[256];
                GetKeyboardState(keyState);
                byte f = 1;
                if ((keyState[144] & f) == f) numlock = true; else
numlock = false;

                if ((keyState[20] & f) == f) capslock = true; else
capslock = false;

                if ((keyState[145] & f) == f) scrolllock = true; else
scrolllock = false;




                // check this ..vaise this works


                f = 0x80  ;


                if ((keyState[(int)Keys.LControlKey] & f) == f) lctrl =
true; else lctrl = false;

                if ((keyState[(int)Keys.RControlKey] & f) == f) rctrl =
true; else rctrl = false;

                if ((keyState[(int)Keys.LWin] & f) == f) lwin = true;
else lwin = false;

                if ((keyState[(int)Keys.RWin] & f) == f) rwin = true;
else rwin = false;

                if ((keyState[(int)Keys.LMenu] & f) == f) lmenu = true;
else lmenu = false;

                if ((keyState[(int)Keys.RMenu] & f) == f) rmenu = true;
else rmenu = false;

                if ((keyState[(int)Keys.RShiftKey] & f) == f) rshift =
true; else rshift = false;

                if ((keyState[(int)Keys.LShiftKey] & f) == f) lshift =
true; else lshift = false;

               // if (lshift) MainForm.handle.LogWrite("numlock");
```

Basically we isolated the left ctrl key as our command key, altering it’s functionality completely. Keys pressed in combination with the Lctrl would provide varied accessibility information and information about various cached events. The scroll lock was used to toggle voice feedback.

The keyboard also served as the control panel of Sonique, with various command keys deployed for orthogonal and centralized deployment of all of the software’s capabilities.

Windows Accessibility at OS and Application levels

Both mouse and keyboard were used to attain accessibility information which would be necessary for the visually impaired to navigate properly.

For this purpose, ***Microsoft Active Accessibility*** was utilized.

The main idea behind Active Accessibility is to provide the functionality to access UI elements programmatically to get information about or manipulate these elements. UI elements that support this functionality are called accessible. In most cases this means that a UI element supports the IAccessible interface.

The core functionality in Active Accessibility is provided by OLEACC.DLL. Each time you call a function that returns a pointer to an IAccessible interface corresponding to a particular UI element, OLEACC.DLL verifies whether this element natively supports IAccessible. Native support means that IAccessible is implemented programmatically for this element.

When a UI element doesn't support IAccessible natively, OLEACC.DLL verifies the Windows class name for this element. If this class is a USER or COMCTL32-supported class, OLEACC.DLL creates a proxy that implements IAccessible on behalf of the UI element. Most—but not all—COMCTL32 controls have IAccessible support provided by OLEACC.DLL.

Examples of UI elements that natively support IAccessible are custom controls, owner-drawn, or windowless controls. Since developers who create software that contains these kinds of UI elements also implement the interfaces for these elements, they are also responsible for providing correct support for methods and properties. In practice that means some methods or properties might be implemented improperly or not at all.

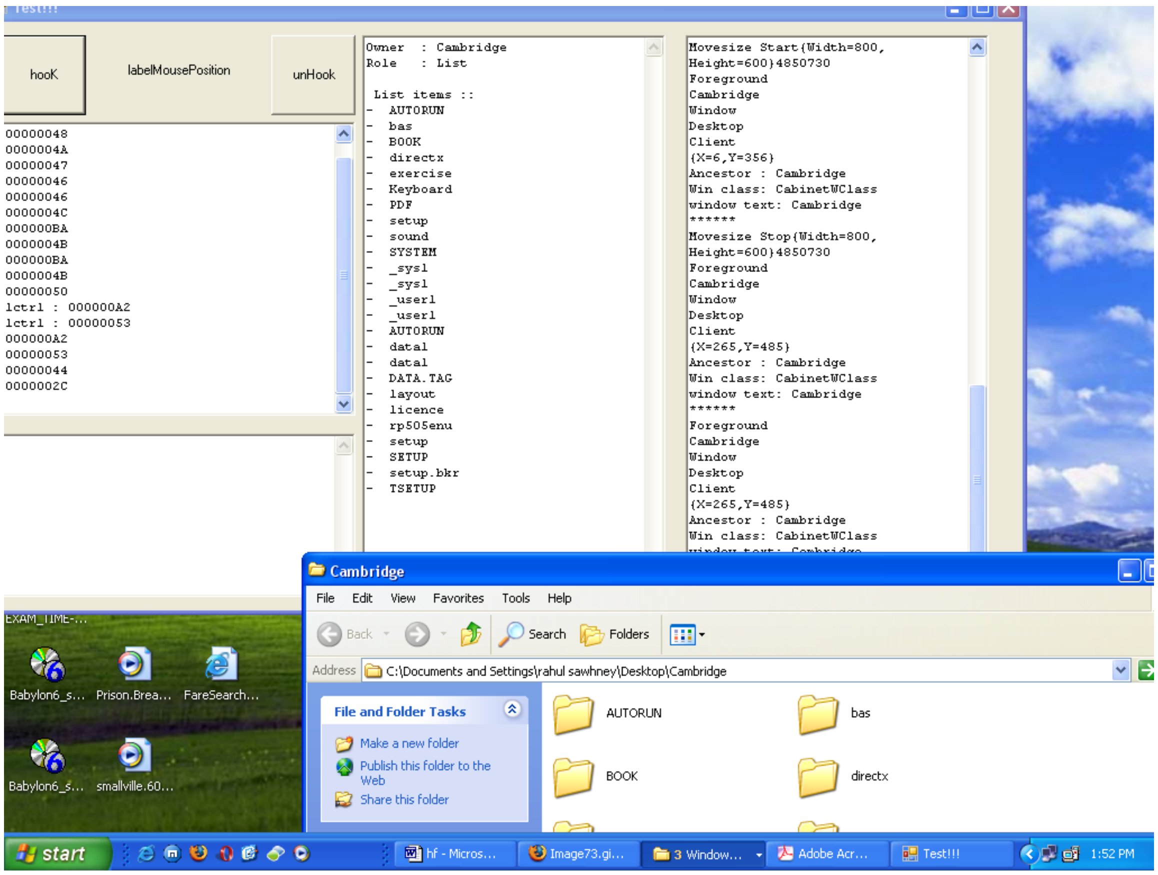
This also means that a developer who implements the interface defines its properties, such as the name and role.

If a UI element doesn't support IAccessible natively and OLEACC.DLL doesn't recognize its class name as supported, OLEACC.DLL creates a default proxy that provides minimal HWND-based IAccessible support, such as location and whether the window is enabled and visible. The default proxy doesn't provide any control-specific information.

Since Active Accessibility is a relatively new technology, it has some flaws in practical use. Most problems arise because there are many controls with only partial support—or no support at all—for Active Accessibility

Sonique has made use of MSAA to make the desktop accessible to the visually impaired. Accessibility information about the UI element under the mouse cursor is fully extracted. In fact, names of all list items in a folder are cached as soon as the cursor reaches a listview UI element (basically a folder). Information like, the name of the current taskswitched item (alt+tab) , list of selected items is also extracted.

In circumstances where MSAA fails to provide us with information, direct Win32 based method were used. In some cases, the techniques employed were specific to current version of Windows. In some cases the Win32 methods had a lot of bugs. And in some cases, there were only legacy MFC methods. (MFC is Microsoft Foundation Classes, a Windows framework used in previous versions of Windows, but now present only as a legacy one for backward compatibility). So, it was only after a lot of experimentation, that we were able to perfectly extract the desired information from Windows.



```
00000048
0000004A
00000047
00000046
00000046
0000004C
000000BA
0000004B
000000BA
0000004B
00000050
lctrl : 000000A2
lctrl : 00000053
000000A2
00000053
00000044
0000002C
```

```
Owner  : Cambridge
Role   : List

 List items ::
-  AUTORUN
-  bas
-  BOOK
-  directx
-  exercise
-  Keyboard
-  PDF
-  setup
-  sound
-  SYSTEM
-  _sys1
-  _sys1
-  _user1
-  _user1
-  AUTORUN
-  data1
-  data1
-  DATA.TAG
-  layout
-  licence
-  rp505enu
-  setup
-  SETUP
-  setup.bkr
-  TSETUP
```

```
Movesize Start{Width=800,
Height=600}4850730
Foreground
Cambridge
Window
Desktop
Client
{X=6,Y=356}
Ancestor : Cambridge
Win class: CabinetWClass
window text: Cambridge
******
Movesize Stop{Width=800,
Height=600}4850730
Foreground
Cambridge
Window
Desktop
Client
{X=265,Y=485}
Ancestor : Cambridge
Win class: CabinetWClass
window text: Cambridge
******
Foreground
Cambridge
Window
Desktop
Client
{X=265,Y=485}
Ancestor : Cambridge
Win class: CabinetWClass
```

Our work on accessibility was original. Although keyboard based screen readers are known to exist, we are yet to find a solution which deploys both mouse and keyboard to attain accessibility information in a manner Sonique does.

The problems we faced were numerous. MSAA has not been used much by the developer community in general and we did not have any examples or code lines to get us started. And the documentation provided for MSAA is for native Win32 whereas we were coding in .NET. Then we had to devise a system which would be make efficient use of processing resources as there would always be some new information waiting to be accessed.

Accessibility of commonly used applications was also increased by deploying an interactive and intuitive menu called ***in-focus menu*** on top of them. Basically, the most commonly used commands were abstracted and presented in a very accessible format.

For experiment purposes we developed the in-focus for Windows Media Player and Explorer.

Frameworks & Platforms used & reasoning

The front end of in-focus was developed on Windows Presentation Foundation [WPF], a recent Microsoft framework for enhanced visual and interactive experience.

All the accessibility features implemented via user interfaces were built on the **.NET 3.0 Framework**[1]. During the initial states of development, .NET 3.0 was called WinFX. We made extensive use of the ***Windows Presentation Foundation*** (or WPF), which is the graphical subsystem of .NET 3.0 Framework.

The reason for choosing to work with WPF is that it provides a consistent programming model for building applications and provides a clear separation between the UI and the business logic. WPF application can be deployed on the desktop or hosted in a web browser. It also enables richer control, design, and development of the visual aspects of Windows programs. It aims to unify a host of application services: user interface, 2D and 3D drawing, fixed and adaptive documents, vector graphics, raster graphics, animation, data binding, audio and video. Besides this, WPF introduces a new language known as eXtensible Application Markup Language (XAML), which is based on XML. Using XAML to develop user interfaces also allows for separation of model and view; this is generally considered a good architectural principle. In XAML, every element maps onto a class in the underlying API, and the attributes are set as properties on the instantiated classes.

Rationale behind the design & development of in-focus Menu

"What is in-focus menu"

The in-Focus Menu is a Microsoft Active Accessibility driven *radial context menu* system that presents the user with the most relevant functions in the current window context. For instance, when the user is using Windows Media Player on his PocketPC, by touching the stylus on the screen, he/she would be presented with a high-contrast radial menu centered at that point. This radial menu will be consisting of the most relevant commands that he would need to use in that program's context. As the user moves the stylus around the point where he first clicked and moves over the different

[1] The **Microsoft .NET Framework** is a software component which can be added to the Microsoft Windows operating system. It provides a large body of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework.

segmented options in the menu, in-Focus will speak out the description of the option in focus. The user can also navigate the radial menu of in-Focus using the direction keys on the Pocket PC and use the central Action button to execute the option in focus. The radial menu is coupled along with a sub-vertical menu as and when required. Such a vertical listing would occur in scenarios like the play-list content in Windows Media Player or a listing of hyperlinks that are present in a webpage being browsed by the user.

The sub-vertical menu can also be navigated by using the up and down direction keys on the Pocket PC and it will speak out the option that is selected. Subtle audio effects has also been included in the design of in-Focus to let the user know of relative position of options in the list and to know when he has reached the end of a list or makes a selection. The in-focus menu has been designed in high contrast so that it would be useful also for a user who has partial visual impairment and can make use of whatever little sight capabilities he has.

"in-focus menu – Structure : why & how"

During the development of in-focus menu, a lot of ergonomics rules had to be kept in mind, with special emphasis on the visually impaired user. We made use of one of the most fundamental laws of ergonomics in user-interfaces, which is called the ***Fitt's Law.*** In ergonomics, Fitts' law is a model of human movement, predicting the time required to rapidly move from a starting position to a final target area, as a function of the distance to the target and the size of the target. Fitts' law is used to model the act of pointing, both in the real world, for example, with a hand or finger and on computers, for example, with a ***mouse/stylus***.

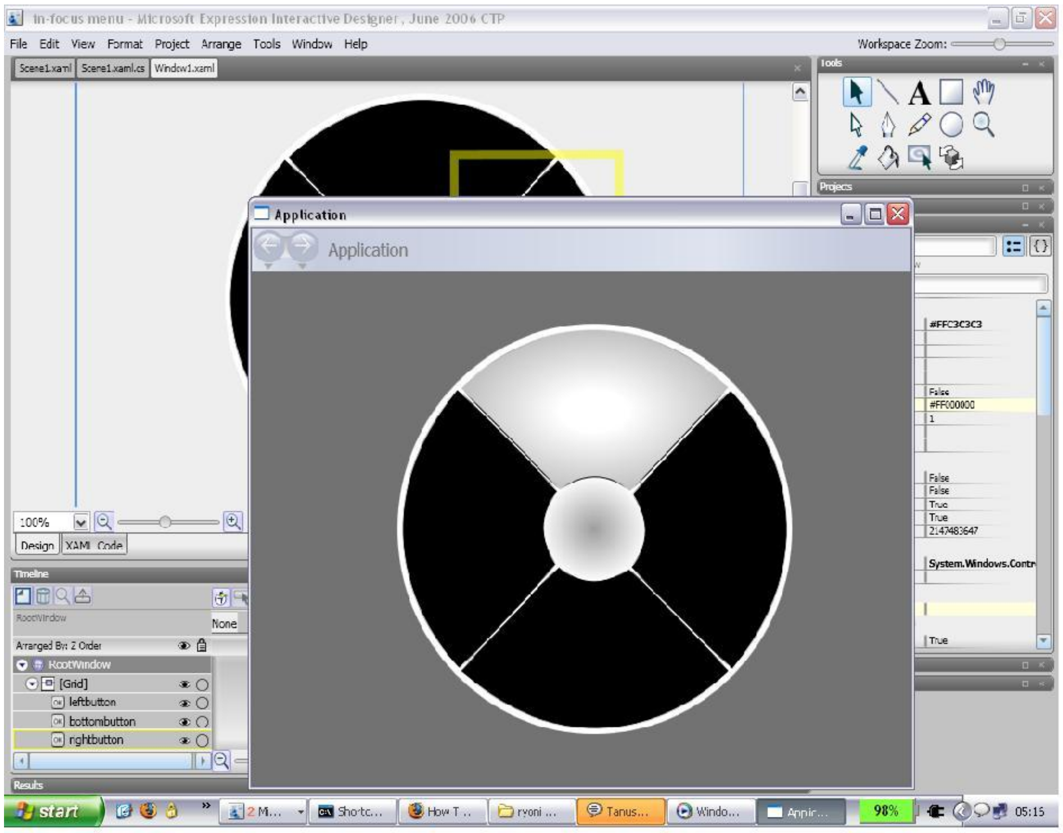
Since the advent of graphical user interfaces, Fitts' law has been applied to tasks where the user must position the mouse cursor over an on-screen target, such as a button or other widget. Fitts' law can model both point-and-click and drag-and-drop actions. Some of the points related to Fitt's Law that we learnt were:

- It applies only to movement in a single dimension and not to movement in two dimensions.
- It describes untrained movements, not movements that are executed after months or years of practice.
- Buttons and other widgets to be selected in GUIs should be a reasonable size; it is very difficult to click on small ones.

And most importantly,

- ***Pie menu*** items typically are selected faster and have a lower error rate than linear menu items, for two reasons: because pie menu items are all the same, small distance from the centre of the menu; and because their wedge-shaped target areas (which usually extend to the edge of the screen) are very large.

Keeping the above principles in mind, we designed the in-focus menu with a structure as shown in the diagram below:



(Fig 1: Work-in-Progress – Structure of in-focus Menu)

In addition to the above guidelines that we discovered and adhered to, it was also necessary to optimize this accessibility solution for the *partially visually impaired* people also. Hence it was very necessary to ensure that the colors used in making this plug-in interface were of a *high contrast-low contrast* combination so as to help in easier understanding for the specific target audience.

It was also necessary to provide sound cues while the visually impaired person would use the specialized in-focus menu. For instance, there would be sound notifications to let him know which button/widget is he currently on, and what are the actions associated with it.

For the back-end of infocus, we tried out several approaches.

One of the ways was to hack into the message interaction of the given application with the underlying OS, in our case, Windows. And then interact with the application by sending similar messages from Sonique.

The other approach we tried was to emulate clicks over the application by quickly moving mouse cursor from one command button to the other in the desired sequence. Since all of that would be happening programmatically, the process would be pretty quick (but not as quick as in the first case).

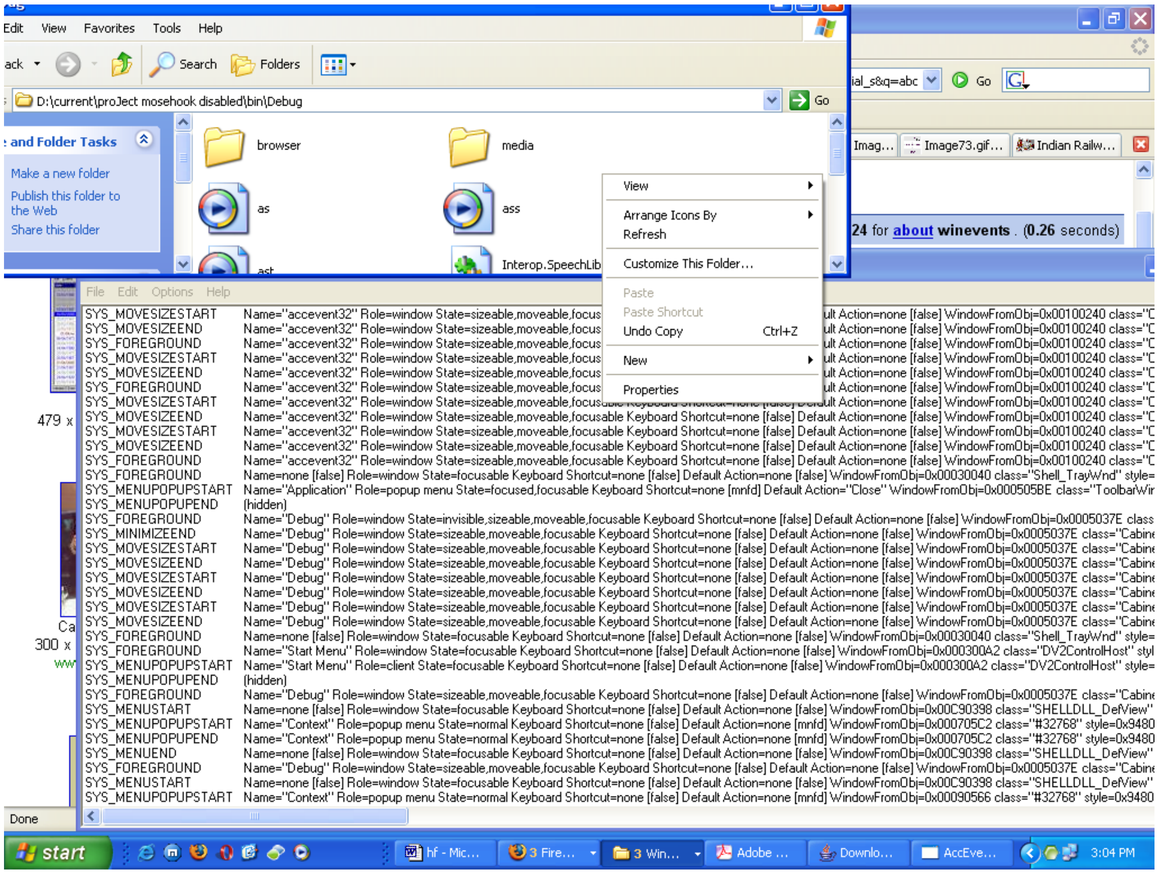
Another approach we tried was to, to utilize the accessibility interface implemented by the application.Though as we know, the IAccessible interface is usually partially implemented.

Another thing we tried was to utilize the API provided by the application and control the application through it.

We were successful in all the above strategies. But we chose the last option because of the degree of control it granted us over the application in question i.e.Windows Media Player.

Windows Event Management System

When we initially designed sonique , windows event handling was not in the immediate scheme of things. But it soon became obvious that sonique was grossly incomplete without a proper event management system which would interact with windows system and gather all possible information(windows events) of use from it. The event information would then be processed , quickly , and presented to the user in a as intuitive and unobtrusive manner as possible , Windows is a GUI based system. An intrinsic part of any interactive system is a feedback mechanism , an acknowledging system , which informs the user of the status of the ongoing interaction. Such feedback occurs subtly, or inconspicuously for us , people with vision. But for the visually impaired, that feedback (most of which is visual) is absent. Thus it is imperative to keep a tab on the ongoing interaction with the OS and provide necessary auditory feedback to the user.



```
SYS_MOVESIZESTART    Name="accevent32" Role=window State=sizeable,moveable,focus    ult Action=none [false] WindowFromObj=0x00100240 class="C
SYS_MOVESIZEEND      Name="accevent32" Role=window State=sizeable,moveable,focus    ult Action=none [false] WindowFromObj=0x00100240 class="C
SYS_FOREGROUND       Name="accevent32" Role=window State=sizeable,moveable,focus    ult Action=none [false] WindowFromObj=0x00100240 class="C
SYS_MOVESIZESTART    Name="accevent32" Role=window State=sizeable,moveable,focus    ult Action=none [false] WindowFromObj=0x00100240 class="C
SYS_MOVESIZEEND      Name="accevent32" Role=window State=sizeable,moveable,focus    ult Action=none [false] WindowFromObj=0x00100240 class="C
SYS_FOREGROUND       Name="accevent32" Role=window State=sizeable,moveable,focus    ult Action=none [false] WindowFromObj=0x00100240 class="C
SYS_MOVESIZESTART    Name="accevent32" Role=window State=sizeable,moveable,focus    ult Action=none [false] WindowFromObj=0x00100240 class="C
SYS_MOVESIZEEND      Name="accevent32" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x00100240 class="C
SYS_MOVESIZESTART    Name="accevent32" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x00100240 class="C
SYS_MOVESIZEEND      Name="accevent32" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x00100240 class="C
SYS_FOREGROUND       Name="accevent32" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x00100240 class="C
SYS_FOREGROUND       Name=none [false] Role=window State=focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x00030040 class="Shell_TrayWnd" style=
SYS_MENUPOPUPSTART   Name="Application" Role=popup menu State=focused,focusable Keyboard Shortcut=none [mnfd] Default Action="Close" WindowFromObj=0x000505BE class="ToolbarWir
SYS_MENUPOPUPEND     (hidden)
SYS_FOREGROUND       Name="Debug" Role=window State=invisible,sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class
SYS_MINIMIZEEND      Name="Debug" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class="Cabine
SYS_MOVESIZESTART    Name="Debug" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class="Cabine
SYS_MOVESIZEEND      Name="Debug" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class="Cabine
SYS_MOVESIZESTART    Name="Debug" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class="Cabine
SYS_MOVESIZEEND      Name="Debug" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class="Cabine
SYS_MOVESIZESTART    Name="Debug" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class="Cabine
SYS_MOVESIZEEND      Name="Debug" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class="Cabine
SYS_FOREGROUND       Name=none [false] Role=window State=focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x00030040 class="Shell_TrayWnd" style=
SYS_FOREGROUND       Name="Start Menu" Role=window State=focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x000300A2 class="DV2ControlHost" styl
SYS_MENUPOPUPSTART   Name="Start Menu" Role=client State=focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x000300A2 class="DV2ControlHost" style=
SYS_MENUPOPUPEND     (hidden)
SYS_FOREGROUND       Name="Debug" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class="Cabine
SYS_MENUSTART        Name=none [false] Role=window State=focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x00C90398 class="SHELLDLL_DefView"
SYS_MENUPOPUPSTART   Name="Context" Role=popup menu State=normal Keyboard Shortcut=none [false] Default Action=none [mnfd] WindowFromObj=0x000705C2 class="#32768" style=0x9480
SYS_MENUPOPUPEND     Name="Context" Role=popup menu State=normal Keyboard Shortcut=none [false] Default Action=none [mnfd] WindowFromObj=0x000705C2 class="#32768" style=0x9480
SYS_MENUEND          Name=none [false] Role=window State=focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x00C90398 class="SHELLDLL_DefView"
SYS_FOREGROUND       Name="Debug" Role=window State=sizeable,moveable,focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x0005037E class="Cabine
SYS_MENUSTART        Name=none [false] Role=window State=focusable Keyboard Shortcut=none [false] Default Action=none [false] WindowFromObj=0x00C90398 class="SHELLDLL_DefView"
SYS_MENUPOPUPSTART   Name="Context" Role=popup menu State=normal Keyboard Shortcut=none [false] Default Action=none [mnfd] WindowFromObj=0x00090566 class="#32768" style=0x9480
```

What we did to accomplish the above , was to ‘hook’   windows event sytem.

Microsoft® Active Accessibility® provides a mechanism called *WinEvents* that allows the operating system and servers to notify clients when an accessible object changes. There are numerous conditions in which a server notifies a client of a change. Each event constant defined by Active Accessibility describes a condition about which a client is notified. For example, WinEvents can signal:

• When an object is created or destroyed.

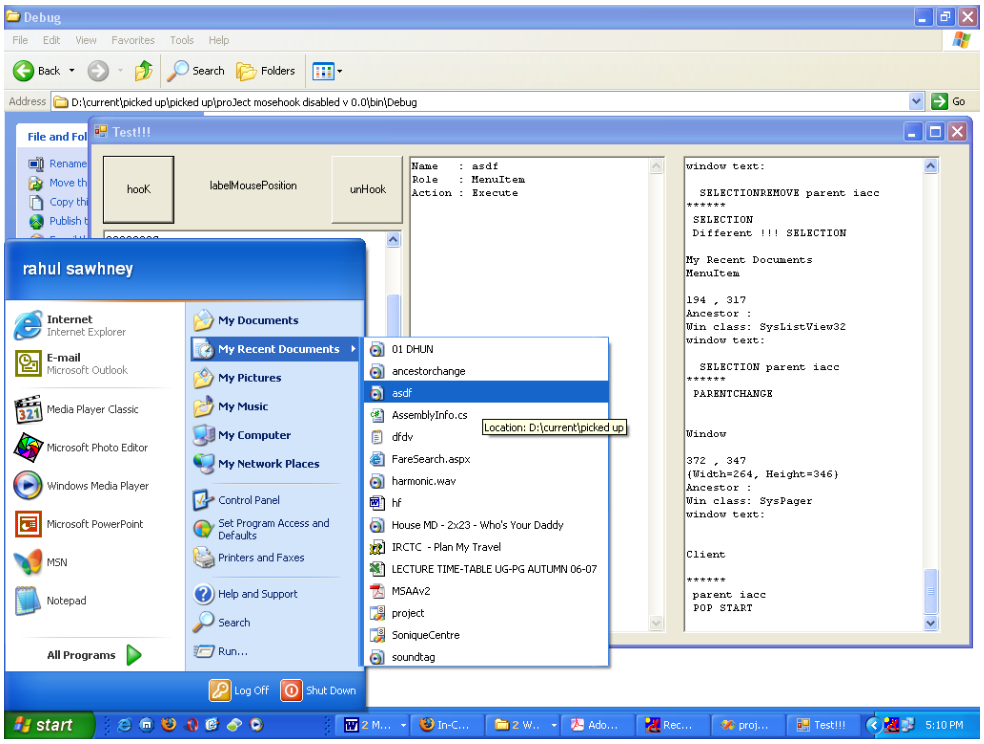
• When an object receives or loses focus.

• When an object's state or location changes.

• When any of an object's properties change.

Problem with WinEvents is that it is not implemented fully and properly. Much of the underlying reasons which fire a particular winevent are unclear. Windows event notification system is incomplete and quite a few functions are buggy. Thus we faced a lot of problems in trapping the relevant events . Usually several event notifications would happen simultaneously. There would be several dummy notifications too. There were several other issues like different events sending similar notifications , a single event (externally ,that is) sending several notifications .

We , eventually , got around these problems by caching all the information and developing a rule system based on a lot of experimentation and observation of events of interest. All possible information about the events and their parents were observed to find the resolution factors .



```
Name   : asdf
Role   : MenuItem
Action : Execute
```

```
window text:

  SELECTIONREMOVE parent iacc
******
 SELECTION
 Different !!! SELECTION

My Recent Documents
MenuItem

194 , 317
Ancestor :
Win class: SysListView32
window text:

  SELECTION parent iacc
******
 PARENTCHANGE


Window

372 , 347
{Width=264, Height=346}
Ancestor :
Win class: SysPager
window text:


Client

******
 parent iacc
 POP START
```

Sound Feedback System

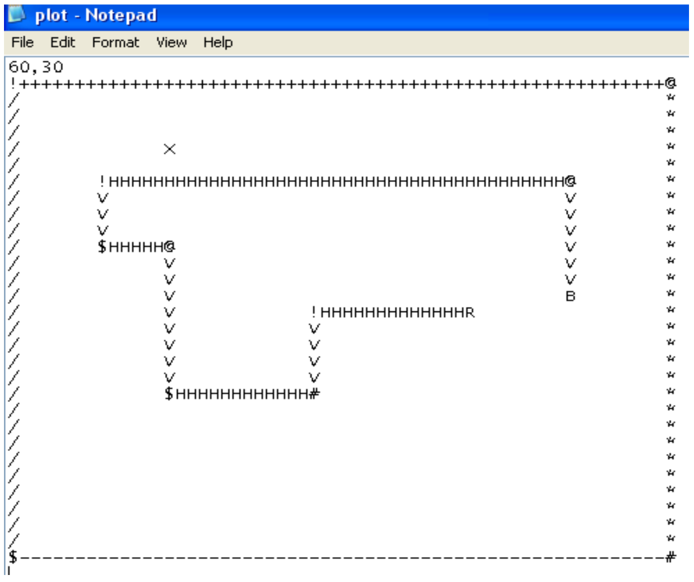
It would not be wrong to say that sonique is all about sounds and feedbacks.   For whatever underlying processing that takes place there is only sound / voice   auditory feed that puts forth the information to the user.

We utilized Microsoft Direct Sound for generating sound feedback and Microsoft Speech Engine for voice feedback.

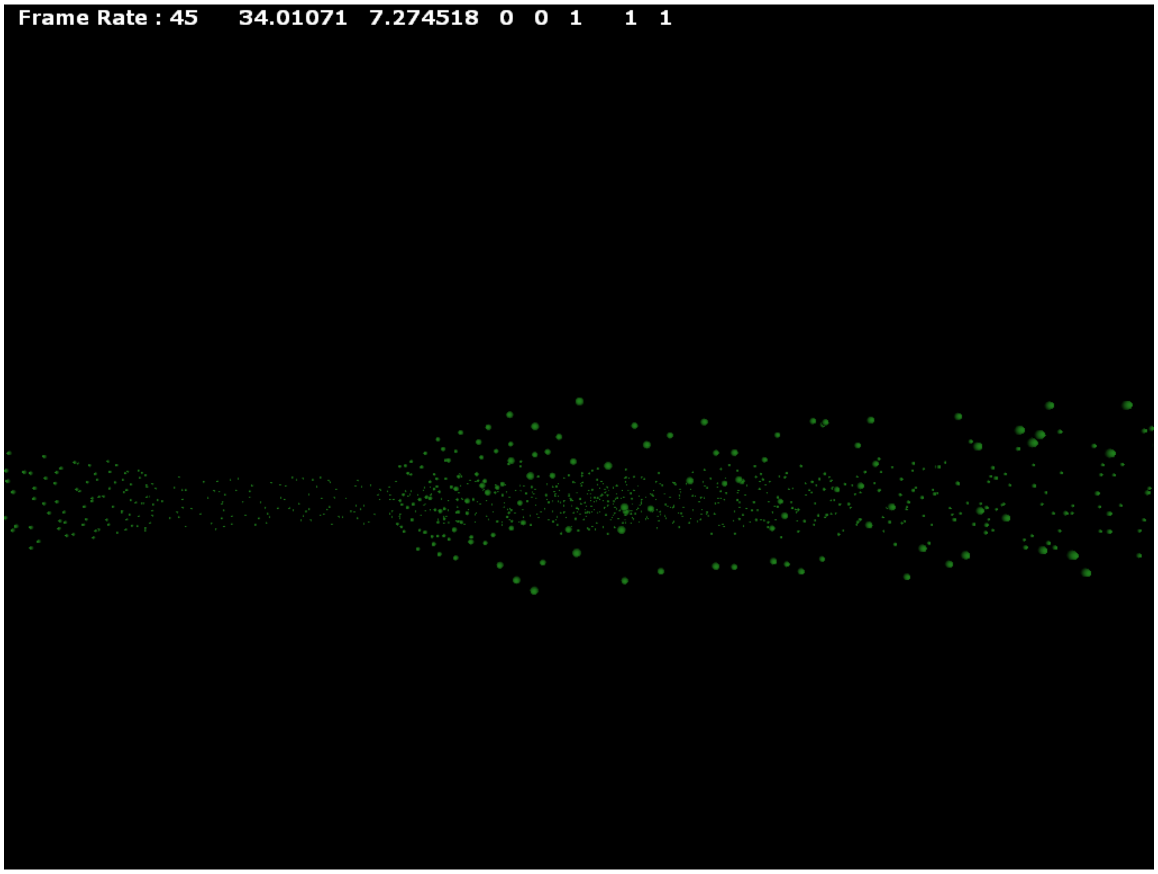
Sonic Map

We visualized ‘sonic map’ as a virtual space completely expressed and realized through sound. What we intended to do was , to translate the data received from sonique’s real time navigation system into a virtual three dimensional space completely expressed though sound and voice tags. Basically, we intended to virtualise the real world , the user’s surroundings , so that the environment and useful information added to it by the user , in the form of voice tags , could be utilized by other sonique users.. Basically , a map built for the blind by the blind.

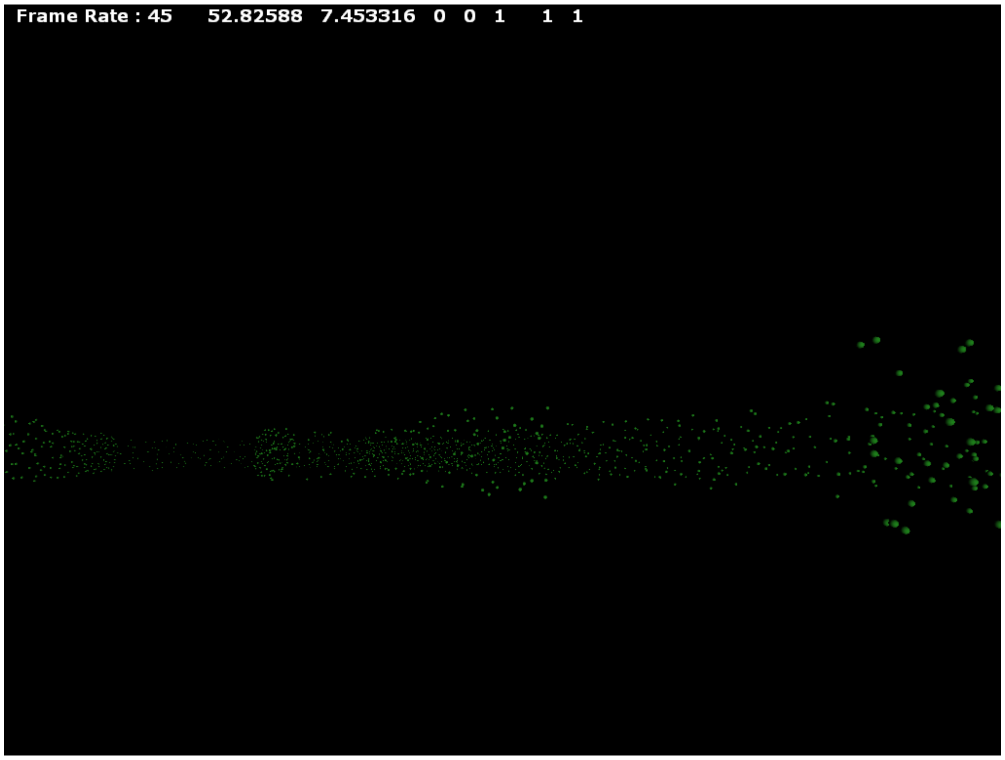
We utilized dummy data to generate the map . In order to add flexibility in map design and allow us to experiment more extensively , the map was generated from a text file which held the the map outline. The map would then be populated with spheres (representing point data)   generated at random points based on the outline provided by the text file.



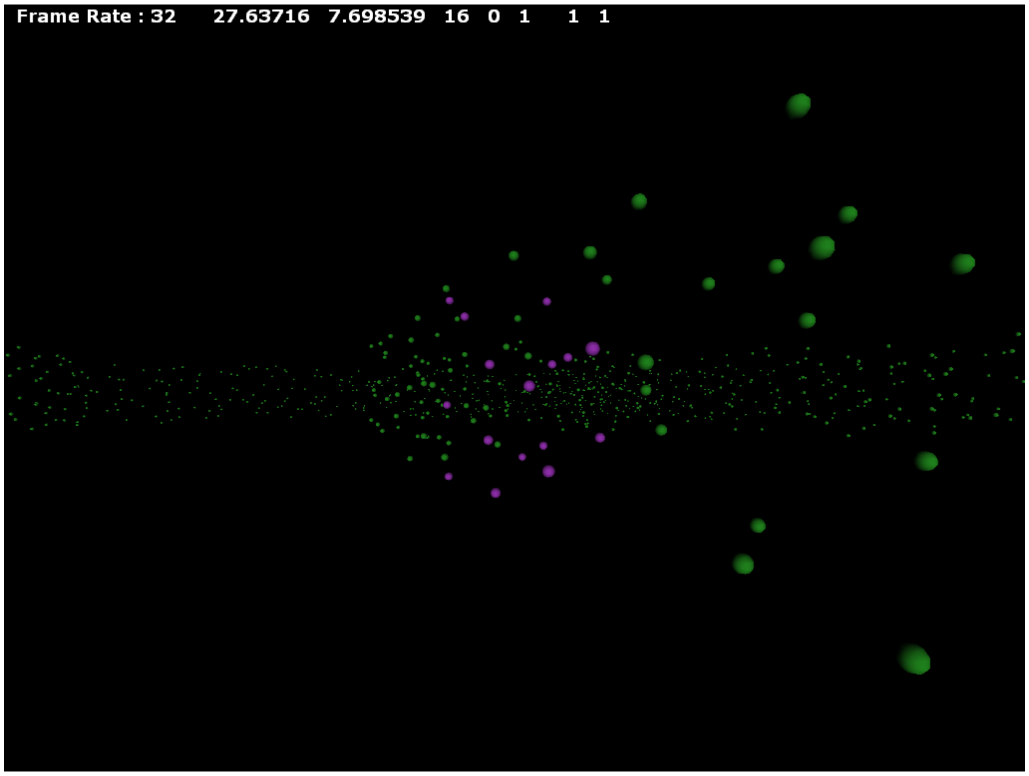
```
60,30
!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++@
/                                                          *
/                                                          *
/                                                          *
/             X                                            *
/                                                          *
/       !HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH@        *
/       V                                         V        *
/       V                                         V        *
/       V                                         V        *
/       $HHHHH@                                   V        *
/             V                                   V        *
/             V                                   V        *
/             V                                   B        *
/             V            !HHHHHHHHHHHHHR                 *
/             V            V                               *
/             V            V                               *
/             V            V                               *
/             V            V                               *
/             $HHHHHHHHHHHH#                               *
/                                                          *
/                                                          *
/                                                          *
/                                                          *
/                                                          *
/                                                          *
/                                                          *
/                                                          *
/                                                          *
$----------------------------------------------------------#
```


```
Frame Rate : 45     34.01071  7.274518  0  0  1     1  1
```

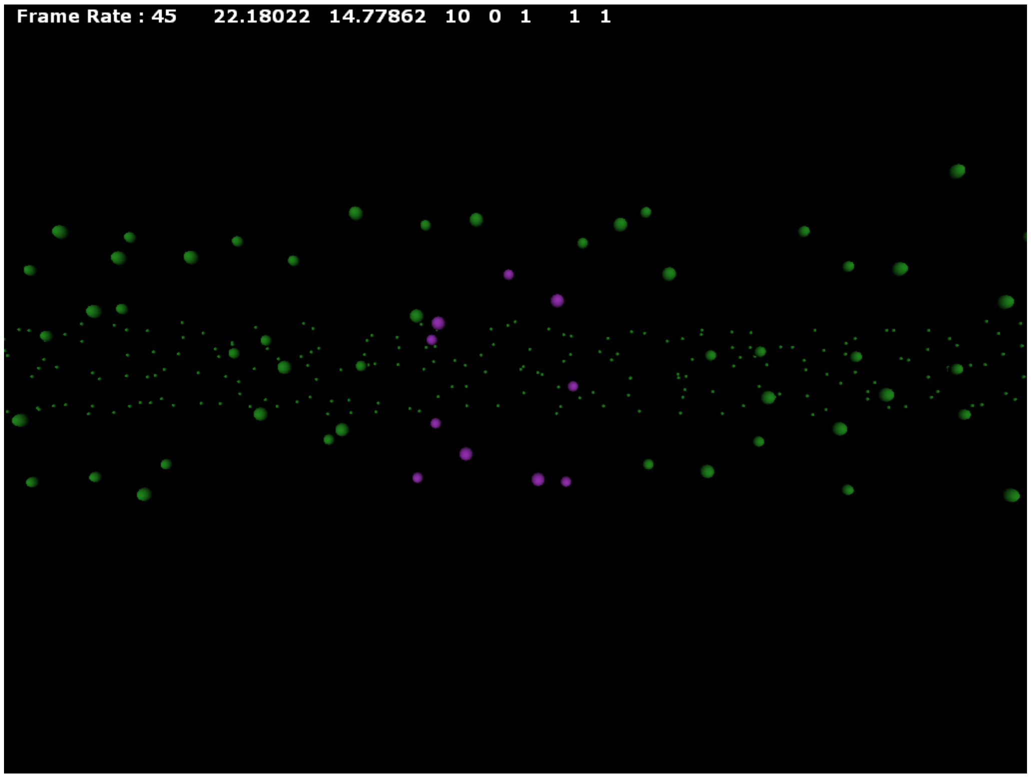
Attempt to make it with textures.



```
Frame Rate : 45     52.82588  7.453316  0  0  1    1  1
```



```
Frame Rate : 32     27.63716  7.698539  16  0  1    1  1
```



```
Frame Rate : 45    22.18022  14.77862  10  0  1    1  1
```

Software on the Pocket PC

For the development of Sonique, it was necessary for the system running the software to be portable and mobile so as to enable anytime-anywhere access to places and information for the visually impaired user. So, to demonstrate the capabilities of the best accessibility software that could be written for a mobile platform, we chose to use a Pocket PC running on Windows Mobile 5.0 operating system.

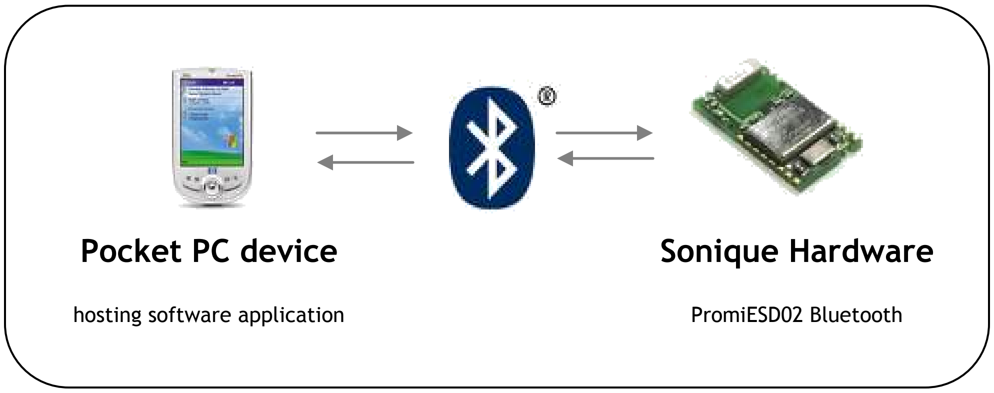
The reasons to choose this platform and operating device are manifold:

- **Windows Mobile** is a compact operating system combined with a suite of basic applications for mobile devices based on the Microsoft Win32 API.
- It comes preloaded with the prerequisite software to properly demonstrate the capabilities of Sonique in terms of different application domains like media player, internet browsing, voice command, messaging etc.
- It can also run programs written for wireless data transmission to the hardware module of Sonique, via Bluetooth®.
- The operating system is able to support frameworks like .NET Compact Framework (.NET CF) which is very essential for designing software solutions faster and better.
    - .NET CF is a version of the .NET framework that is designed to run on mobile devices like PDAs and SmartPhones. To make it more compatible with small devices such as these which have inherent memory constraints, .NET CF uses only some of the same class libraries as the full .NET framework.
    - Even though .NET CF did not fully address our requirements for designing the total software system of Sonique, we were able to come up with alternative frameworks to augment the .NET CF to provide seamless software design capabilities. The reasons for the choice and implementation of extra frameworks are illustrated later with problem scenarios.

Bluetooth Communication Module

The first module that was written and implemented on the Pocket PC was the Bluetooth communication module. For this, programs were written to enable wireless communication between the Pocket PC and the Bluetooth chip of the hardware module of Sonique. Some of the steps involved in this process was detection of hardware, pairing of devices, collection of data, packaging the data, and finally transfer of data.

We made use of the Microsoft Bluetooth Stack for programming purposes to enable connection between the Bluetooth-enabled Pocket PC and the Bluetooth chip of our custom-built hardware.



One of the most important things we found out during development of this module was that the .NET Compact Framework did not contain pre-defined class libraries to write code for applications that makes use of Bluetooth protocols.

Hence, to implement Bluetooth communication, the whole data transfer process had to be abstracted in a ***serial port format***, with the creation of ***virtual COM ports*** which would send/receive data to/from the hardware device.

This meant that we had to make use of a framework outside of the .NET Compact Framework because it did not allow easy implementation of serial port programming. The alternative that we discovered was a framework called **OpenNet Compact Framework (OpenNet CF)** which enriches and extends the already existing .NET CF by providing a rich and useful set of classes and controls that were previously unavailable.

With the Microsoft Bluetooth Stack, we had to setup mutual incoming and outgoing ports on the two Bluetooth enabled devices. For the Bluetooth chip on the hardware end, it had an inbuilt stack which could handle this. This meant that we had to handle the port specifications only on the side of the Pocket PC.

To create a Bluetooth serial connection, we have to create and then open an incoming port (eg. COM6) Pocket PC to allow data to be sent to it from another device. Initially, this port is also used for detection of the other Bluetooth device in range (here, it is the PromiESD02 chip). We observed that there is no need to explicitly define a target device as the incoming port will detect and pair with any existing BT device in range.

Using the MS BT stack ensured that multiple connections would not be accepted simultaneously over a serial connection by the Pocket PC. This would come useful in scenarios where a lot of Bluetooth devices can be found in range, and so the danger of cross-interaction between the devices in question is not a problem because all of the devices are paired and send/receive data only to each other’s partner devices.

Interface Development Issues on Pocket PC

The initial plan during the development of Sonique was to have the complete software system running on a Pocket PC. But there were a few issues to be tackled, and sometimes due to the incapability of the programming environment with regard to the operating system some issues could not be solved in time. Hence, some of the features that we had envisioned for Sonique to be running on a mobile device, now run on a full-fledged PC which has all kinds of graphics and sound options.

To address the problem that we faced more specifically, it was the *task of capturing the stylus contact information* which we could not achieve on the Pocket PC.

In a normal Pocket PC with default Windows Mobile 5.0 operating system settings, any contact of stylus with active area of the screen is captured by the OS itself, and then processed.

For the implementation of Sonic Grid on the Pocket PC, we had to get information regarding the co-ordinate points of the contact location of the stylus on the screen to map it to the underlying transparent grid which produces the corresponding tone. The problem we discovered was that Windows Mobile does not allow us to capture stylus coordinate information and then pre-process it before the location information is processed by the OS itself. In our case, what we ideally required was to trap the coordinate information, block it being sent to the OS, and then using the x-y screen coordinate information, perform the action that the visually impaired user intended to do. The layer of abstraction that we required between the OS and the touch-screen driver could not be implemented because it is not possible to change that aspect once the OS is already built into the Pocket PC.

To resolve this problem and to achieve the desired features in the Pocket PC itself, we tried rigorously to come up with a solution. The other option that we came up with was the use of **Windows CE Platform Builder**.

To explain this approach, it is necessary to understand what exactly Windows CE is. **Windows CE** (sometimes abbreviated **WinCE**) is a variation of Microsoft's Windows operating system for minimalistic computers and embedded systems. Windows CE is a distinctly different kernel, rather than a "trimmed down" version of desktop Windows. Windows CE is a modular/componentized operating system that serves as the foundation of several classes of devices. With the help of Platform Builder, it was possible to build an operating system from scratch choosing different components that we specifically required from the large pool of Windows CE components.

This would ultimately mean that we could in fact insert a layer of abstraction between the OS and the touch-screen driver to obtain information of stylus location, and then process it without handing over that information to the OS. But since we were not familiar with Platform Builder and due to time constraints, we did not implement this solution, and hence reverted to implementing Sonic Grid on a full-fledged PC/laptop.

It was also understood that in the event of using Platform Builder CE, it would be best done if we were trying to build a custom embedded device running customized OS. This would give us complete freedom to decide what OS goes into the hardware being built, and the level of control the programmer has over the different components in the software foundation system. But since our current use was limited to a Pocket PC, we kept this solution as an option for a long-term goal of designing an embedded hardware for this purpose.

Blindness, navigation and aural substitution

The current ocular civilization, impersonal and external has over thousands of years, won over our benign, spirited auditory predecessors. Philosopher of history Oswald Spengler has described us ocular moderns, as the aggressive predators. Through our vision we distance ourselves from our environment, and treat is as an intrinsically collection of hostile objects, targets and victims. The increased dependence on our visual sense has created a background which makes it impossible for the visually abled to empathize with the blind. It is therefore important to explore identity and meaning in the world of the visually disabled if a system is to be developed that can augment their capabilities. These studies we hope have provided a foundation for us to better interpret the needs and expectations of our user base. As an engineer of an interaction system it is important we understand the user to set up appropriate constraints.

Research relating to the understanding and mental models of the blind require an early distinction between the congenitally blind and the others. The extraordinary autobiographical account of John Hull (1990) provides an insight. “During the first couple of years of blindness,” he writes, “when I thought about people I knew, they fell into two groups. There were those with faces, and those without faces …. as time went by the proportion of people with no faces increased.” With those whom he knew, there were vivid images, and of those who had no visual memory, were incontinent visual ‘projections’ created by the brain that was suddenly cut off from the normal sensory input.

With time Hull found, that he moved deeper and deeper into what he calls “deep blindness” with less memory and need of, visual images, and more and more the sense of living in a complete world of body sensations, touch, smell, taste and off course hearing. Even though many congenitally and the “deep blind” use visual images and metaphors in speech, they are only metaphors for them. But it is important to note that visual models may be different in the case of the congenitally blind and those disabled later in life.

As a further study of blind children, it is found that they tend to become ‘hyperverbal’, to employ elaborate verbal descriptions instead of visual images, trying to deny, or replace visuality by verbality. This tends to create, the analyst Dorothy Burlingham (1972) thought, to produce a sort of ‘pseudo-visual “false self”, a pretense that the child was seeing when it was not.

Further participant research and writings by the blind such were followed as parts of the research to help us better understand the user. The above studies have influenced the design of the product and provided an impetus for questioning the same at various stages.

In the design of our Interaction System as a sensory aid, we found it important to understand the human Senses, social pressures on the different senses and modality.

Navigation and Sound

The real-world system was designed to help visually impaired navigate better in their environment. To design a modular, pragmatic device we looked at the various tasks that are required to move from one point to the other. It is first required to ascertain the destination and be aware of the current location, to be able to *navigate* in between defined *waypoints* and finally to avoid obstacles and hazards when following a path.

Wayfinding involves

1. sense and evade obstacles and hazards
2. navigate to remote location (next waypoint)

and navigation means

1. to continually place 'self' in relation to a absolute

The visually impaired are at a disadvantage as the feedback they receive from the environment does not allow them to accurately update their position as they move in unfamiliar environments. Assistive technology is developed in two broad categories, to help them avoid obstacles and to provide necessary information to a traveler for wayfinding and navigation.

With the development of technologies such as infrared and RFID, designers have come up with low-power transmitting nodes which transmit information about signs, important locations and junctions to the blind, who receive the information using a receiver device (or a mobile phone). But such systems like Talking Signs (Crandall, Gerry, and Alden, 1993; Loughborough, 1979) can only be used in specific areas such as airports, shopping malls etc, and do not provide much help in wayfinding.

Solutions based on GPS provide wayfinding and navigation information in outdoor environments but the issue of obstacle detection is not addressed. Also in many GPS based solutions the audio interfaces are not properly designed and are primarily speech based which are slow and cumbersome.

Studies done to evaluate auditory display modes and guidance methods (Loomis, Golledge and Klatzky; UC Santa Barbara, Carnegie Mellon 1998) show quantitatively, based on tests the navigational ease and performance by the visually impaired. The study emphasized the need for bearing information (using compass) to the person traveling and also that there is a significant performance increase if virtual spatialzed sound is used to provide the bearing. It is necessary to clarify that the study conducted was done with the support of 10 visually impaired men and thus it does not represent differences (if any) in the responses to the interface based on gender.

The evaluation of the needs of the blind after conversations at the Blind School in Gandhinagar and with Mr. Ranchod Bhai at Blind People's Association a set of requirements were drawn up that the design would try to address. These were primarily connotations of the concerns of the visually impaired in their daily lives.

> The most hazardous are the small holes, negative elevations while walking as the walking stick does not warm them of these.
> High obstacles also pose a similar threat to them as they do not register as obstacles to the walking stick.
> An obvious issue was that they would greatly appreciate if they could 'see' beyond their stick and in a more intuitive manner.
> The visually impaired are concerned about how the visual world perceives them and are aware that strange devices would make them the object of mockery. Thus another concern was that the device should not be blatantly visible and exemplify the fact that they are blind.

The above concerns were central to the design process that led to our solution, even though we can not claim to have addressed all the issues to our satisfaction.

A design that was pondered over at various stages but could not be implemented included sensors that were placed at different angles vertically. Such a design would provide us with the needed information to identify holes, bumps etc while walking. Due to the increase in cost and the reconfiguration of our hardware circuit this option was left to be included after relevant user study was done using the developed prototype with 3 sensors.

Products by the KASPA and the sonic pathfinder group have chosen to place their devices on walking sticks of the blind. Such a model is preferred as it would be natural for the blind and also because it is least visually jarring. But due to miniaturization and the availability of eye-glasses with embedded Bluetooth sound receivers, products such as vOICE have started using head mounted devices. We evaluated the following options for the placement of our sensor device

on the arm
head mounted (on a cap) or as part of eyeglasses
as a waist belt
placed on the blind man’s walking stick

Due to our expectations to include a GPS device and a digital compass to the device, placement on the arm or the stick was considered to be impractical. This was in consideration of the inertial time required for the GPS to get positioning data, and the response time of the digital compass models under consideration. It was found, through discussions with people at the Blind People’s Association that generally the first preference would be to place the device on the walking stick. It was only the youngsters who were more receptive of the fact that they could do away with the walking stick, and move around more freely.

It was agreed upon as a team, in compromise and in response to research papers available on the design of wearable navigational devices by the blind (vOICE, GeorgiaTech’s SWAN) that a final decision regarding the placement of the device can be reached only after a functional prototype is available for testing in with the people. Such tests have not been performed and we are hoping to conduct the same very soon, to get more information on the response to the placement of the device. It was thought essential that at that point we lay down a set of expectations of the user tests such as

requirements for enhancing usability of the feedback device
in terms of audio response
the physical modeling and issues such as weight, angular placement.
concrete pros and cons for the various placements of the device
general feedback on the confidence of the user to the feedback
look at the possibility of placing the device on a walking stick
set up issues that will help in the better design of a training module for the system

Detailed user testing relating to optimize the aural feedback of the device was conducted within constraints of having no prototype, and holidays in the Blind People’s Association. We conducted the study with help from a faculty at the BPA, and feedback in a computer generated virtual environment. (Sonification lab).

A study of auditory displays and minimum attention user interfaces helped us create a basic framework for the development of our aural feedback and for the choice of sounds.

Continuous aural feedback from the environment will provide more information to the user, but it is disturbing for the ear. It also creates problems as the visually impaired are heavily dependant on environmental sounds. Studies have shown that the time between successive bearing beeps should be about 5 sec (Holland, Morse & Gedenryd; Open University, UK). Our audio feedback which involves three sounds in quick succession which will take around 150ms and the default intermittent time between sensing was set to be 6 sec. This timing was not set at the embedded layer, but in the phone program which allows us to allow the user to easily change the timing between successive feedbacks. *It is expected that user tests with the prototype will allow us to further understand the timing that is best suitable in real conditions.*

Of the 4 cardinal points - front, back, left and right – three are easily comprehensible when listening to spatialized audio, the difference between front and back requires additional considerations. In addition, it is generally possible to perceive if a sound source is at some intermediate point, giving at least eight compass points (5 if spatially emulated) . If a better angular precision than eight cardinal points is required then more affordances are needed. The concept of a 'chase tone' (Holland, Morse & Gedenryd; Open University, UK) can be employed to aid in resolution of a direction using sound. A fixed pitch tone appropriately panned is accompanied by another sound called the 'chase tone' which is placed in the same direction its tone is representative of the angular distance to the center. We were using sensors that were fixed in angular space therefore the concept of the chase tone was not required, but it is clear that it is easy for the mind to understand differential tones, and therefore the final sounds that were programmed on the mobile phone used a similar method when representing the distance using change in frequency. A fixed tone base tone remained in the background and changes in an overlaying tone represented the distance of the object sensed.

The study of electronic travel aids (ETA) such as the Navbelt and the AudioGPS system helped us improve our earlier tone based design. These systems used chromatic tone changes rather than linear pitch changes, which are more pleasant sounding. This also led us to later explore for the Sonic Grid Indian 'ragas' and western classical music scales to create amiable soundscapes. Also the timbre of the output sound for the 3 different directional sensors is to be selected based on the timbre and frequency overtones of musical instruments. Such a composition for the sound would also allow for easy contextualization of the system simply by basing the sound on local instruments and musical structures. This area has not been explored fully and we hope that we will be able to create sets of sounds and can test the response of users to locally inspired sounds in contrast to linear tonal gradients.

This is a point of contention as our initial efforts to make the sounds information rich by using beats, linear frequency differences did not give anticipated results. The users do not treat the sound as an informative input but rather discard it as noise. For people who understood the concepts behind the sounds it was easier to follow the 'strange' sounds, but we found it difficult to encourage students at the BPA to try and adapt to the sounds.

The design decision to use a quick succession of sounds representing the distance from different sensors as opposed to testing if it would be possible for people to comprehend a complex of tones providing similar information was led more by implementation issues rather than research. The hardware overhead to use three different sensors at the same time was thought to be very high. It would entail the use of a different microcontroller, addition electronic design to convert the sensor distance into PWM signals. Other issues that would hinder development would be crosstalk between the sensors and need for additional noise cancellation.

We looked at algorithms or electronic systems such as Error Eliminating Rapid Ultrasonic Firing that would help us overcome the above mentioned issues. EERUF allows multiple sonars to fire at rates that are five to ten times faster than those attained with conventional ultrasonic firing methods. At the same time, EERUF reduces the number of erroneous readings by one to two orders of magnitude. (J. Borenstein and Y. Koren; IEEE conf on Robotic Automation, France 1992). An implementation of the same was taken up by the Navbelt developers in 1998. It was found that the algorithm demanded resources that were not suited to our design which was to be based on a 8-bit microcontroller and a mobile phone opposed to the mobile computer used by Navbelt. The obstacle avoidance algorithms were not tested due to time constraints that were intensified because of the burnout of a Bluetooth device.

The issue that the device would cause problems in the perception of environmental sound by the visually impaired needed to be addressed. "The downside of virtual sound … is that blind people use their hearing to pick up subtle environmental sounds, and they also use high frequencies to detect surfaces — it's called echolocation,"... "Because they rely on that to avoid bumping into things, and to be aware of their environment, most blind people are reluctant to consider wearing earphones." (Loomis, Marston et al;

UCSB 2004) The solution was found to be in the type of headphones that should be used in outdoor environments. Designs such as the open-backed headphones or the in-front localization (IFL) headphones are more suited for use with the solution. These headphones are designed not to interfere with the environmental sound and provide better spatialization by allowing for aural crosstalk which is generally absent in headphone listening.

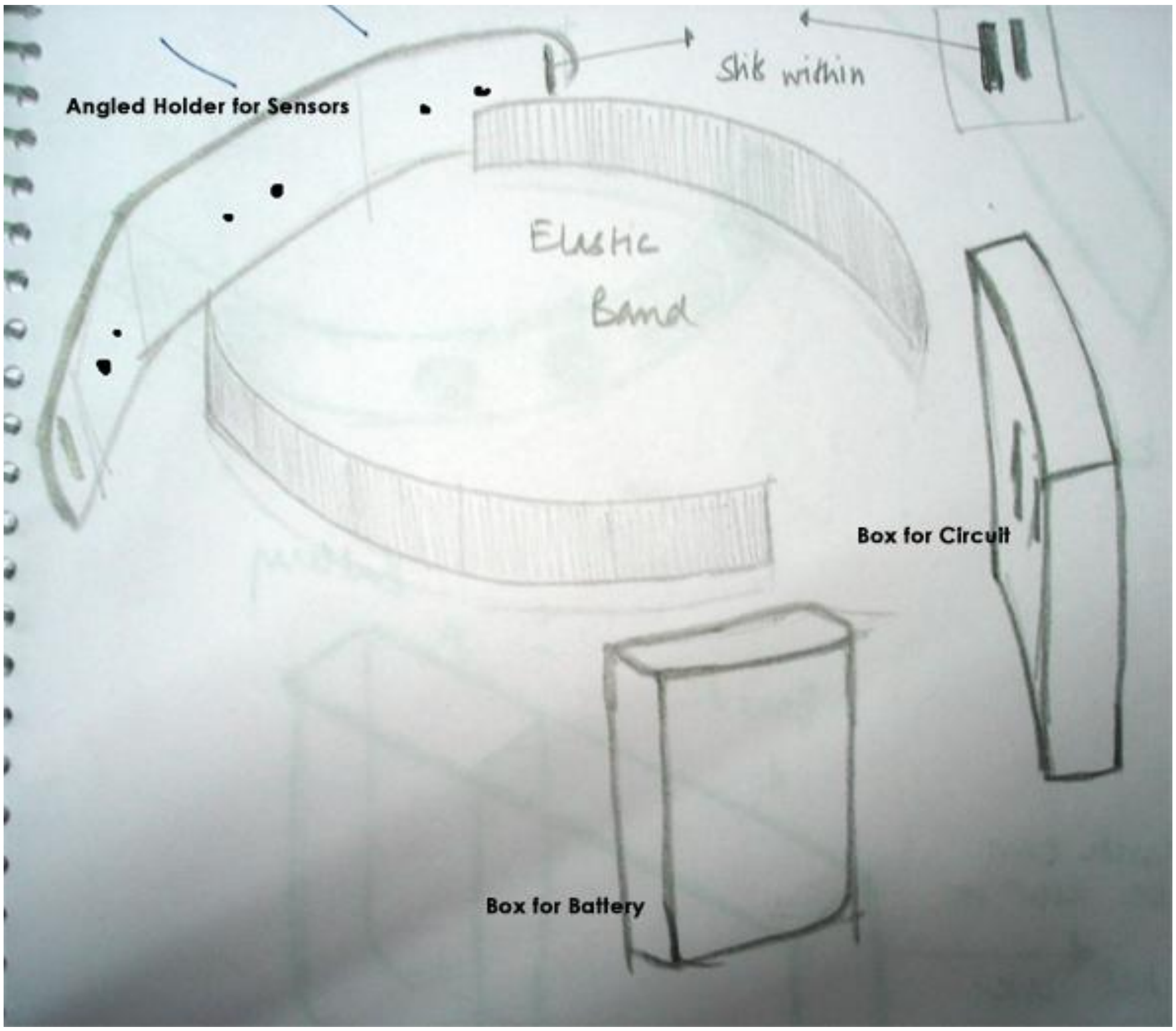
Many people argue against the use of sound based feedback in ETA devices because of the above problem, even though sound is most natural and rich to provide navigation information to the visually disabled. Researchers at the Sonification Lab at GeorgiaTech have developed Bone Conduction Headphones 'Bonephones'. The design sets up sound waves by vibration in the cochlea (inner ear) which is a bone and skips the outer and the middle ear where sound travels through air. Bonephones developed at the sonification lab are able to provide spatial audio to the user without obstructing his ear and the environmental sounds. Spatial audio is created by light timing and intensity differences that are crucial in detecting the location of a sound source. Spatialized audio presentation relies on the ability to present different signals to the two ears, in order to mimic these interaural time and intensity/level differences. True spatialized sound through new software (like Directsound3D) uses head related transfer functions(HRTF) to create 3d audio which places the sound source in relation to the azimuth, elevation and range. These functions are not directly translatable to bonephones and attempts to develop Bone Related Transfer Functions (BRTF) which convert sound presented with HRTF are ongoing.

For the current design we have considered only the azimuth for sound feedback as the mobile phone does not allow enough processing for true 3d spatialization. We have used frequency scale to represent range and stereo panning to place the source in the appropriate direction.

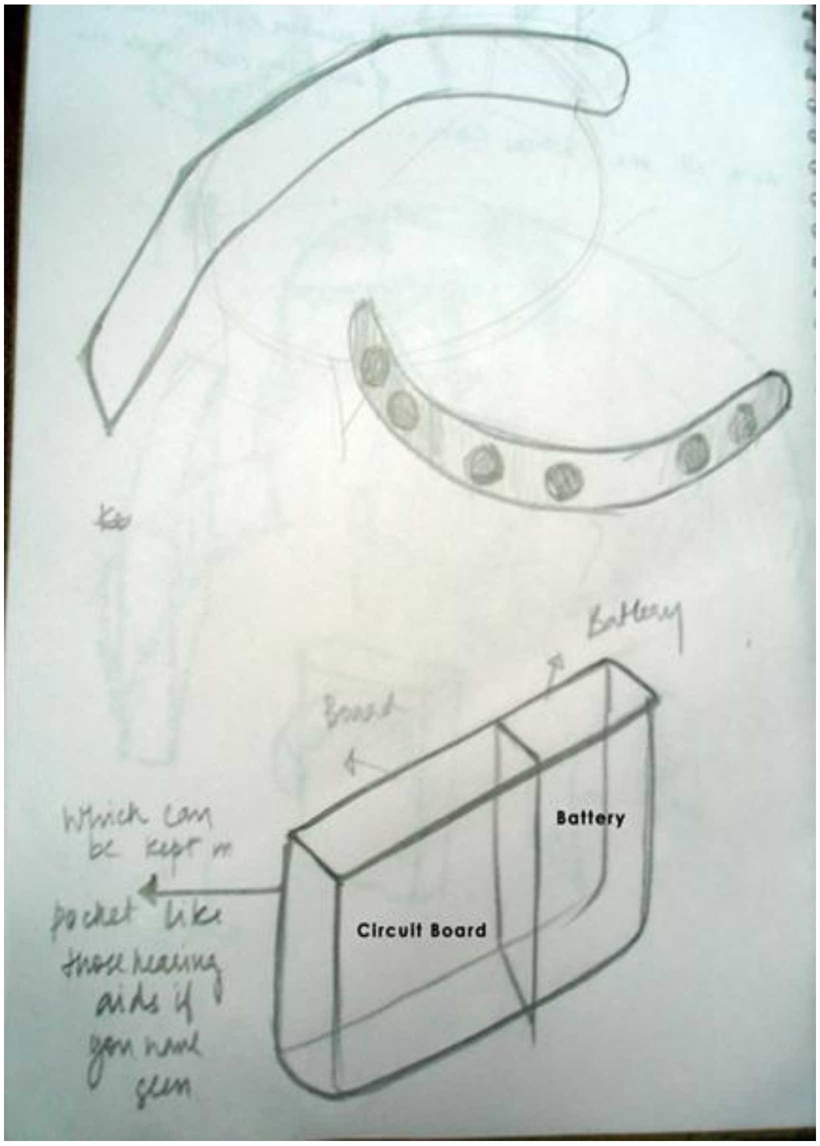
A physical prototype of the device was built to enable us to test our design with users at the BPA with help from Soumya (NID product design student). It was required that the prototype be flexible enough to be used on the head as well as the waist or even attached to the walking stick. It was therefore decided that it would be a belt based design with an enclosing box for the circuit and the battery pack, and angles device to mount the sensors.

Two basic designs were drawn to see the fesability of making them in various materials. The first design modelled the standard hearing aid style, with wires running from the sensor mount to a box which contained the battery and the circuits. It was seen to be the more usable, wires would run from the head/waist to the pocket but it removes the unnessasary burden from the head.

The other design was a burden on the head, but would be independent, with all the equipment resting on the belt. It basic difference between the two was not too much only that the later had separated boxes for battery and the circuit.



It was clear later, due to the big size of the battery pack that the cirvuit would be separate from the battery. The initial expectation of mounting the circuit and battery holders on the elastic belt was also not feasible due to the size.


Battery
Board
Battery
Circuit Board
Which can be kept in pocket like those hearing aids if you have seen

Initially materials such as wood, acrylic and thermocol were considered. Due to acrylic's light weight and ease of cutting the device was constructed from a laser cut 2mm sheet. The cut pieces were glued together to form a box for holding the circuit, battery pack and an angled surface to mount the sensors. To create the angular mount, the edges of the pieces were carefully filed away and then glued. Due to our miscalculation of the power of the battery pack required the final device could not be fit into the box created.

Sonic Grid and User Feedback

The Sonic Grid in an interface enhancement that allows the visually impaired to access the elements of a GUI based system in its intended non-linear spatial representation. The Graphical User Interfaces provide users without disability an easy access to the computer system by providing object metaphors and through a spatial arrangement these interfaces allow us to shift swiftly between tasks and concerns.

In order to facilitate effective use of the spatial pointing devices such as a stylus or a mouse a visually impaired is presented with an invisible and interactive grid layer on the screen, which gives instant sound feedback of the exact position of the pointer on the screen.

The sound identity of a location on screen can be uniquely identified by a combination of some frequency and pan (sideways movement of stylus on screen). Frequency is mapped on the vertical axis of the imaginary grid such that at the bottom of the screen, it is very low and as the pointer moves up, the frequency gradually increases as per the gradient; when the pointer is on the right-most side of the screen, the blind person will hear the frequency via only his right earphone. As he moves the cursor from right to left, the blind user will be able to feel the sound output pan from his right earphone, gradually to both earphones (while in the center of the screen) and then only to his left earphone thus successfully simulating the pan effect.

The basic sounds used with the Sonic Grid range from about 100Hz to 6000Hz as higher frequency sounds are very unpleasant to the ear. The base sounds we are using to test the Sonic Grid are a complex of close frequency which also produce beats, which makes identification in frequency easier. But the variation in the frequency with the pixel shifts on screen is linear, and the change in frequency is calibrated by dividing the screen into equal sized grids.

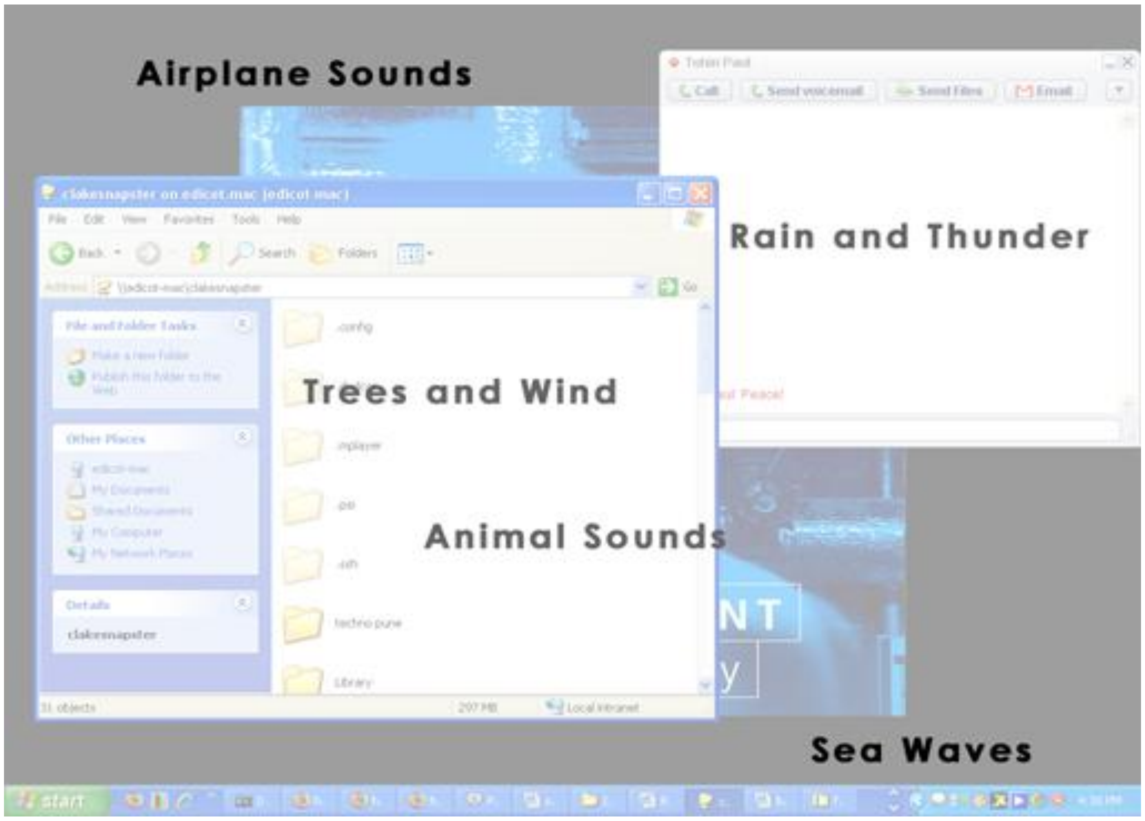
In the tests with students at BPA, Ahmedabad we observed certain problems with the design of the grid such as the fact that it was hard for the users to interpret the sounds as information. Although with practice the sounds can convey the essential information, it is required that the person using the system has a mental model of the GUI. Further probing revealed that spatial training is not a part of the education that visually impaired students in India. With further research into the methods of spatial training by institutes such as BECTA, UK we realized that the sonic grid can help students gain spatial skills, but this would require the development of a training module that would sensitize the blind to the basic terminology of the GUI. Also tactile models of windows and buttons, apart from letting them play audio games such as AudioQuake can help in the understanding of virtual space and sound.

With support of Soumya Mannivanan (student, Product Design; NiD) we explored the possibilities of such improving the sound feedback of our system. A initial list of experimental soundscapes were picked up to be developed

1. Sounds inspired from the environment
   - such as a gradient(intensity and speed) of raindrops falling
   - winds
2. Landscape sound samples for children
3. Sounds inspired from 'ragas' and western classical scales

We are continuing to search for, and trying to build a sound library that can be used to represent the pixel height on the screen. For even a fairly clear representation of the average screen at the resolution 1024x768, with toolbar icon heights approximately 10-15 pixels it would require about 80 sounds which are also in a gradient. It has been shown that difference is the basis of understanding in language and therefore the sounds used should have a general gradient so that the change in position when the mouse is moved.

It is challenging for the visually impaired to start learning computers as it requires a huge amount of memorizing and training. It was also observed that children do not appreciate computers because of the same reasons, we collected sounds samples that represent a landscape to make computers fun for children at the same time making them comfortable with computer use (keyboard and mouse).



Since the above methods were primarily based on sound collections and could not be generated dynamically we looked at the sound octave scales to create pleasant sounding sonic grid feedback.

The western half-notes are separated on a logarithmic scale so to calculate any tone above or below the standard A440 note the following mathematical expression can be used:

$$440 * 2 \wedge (n / 12)$$

where n is the number of half steps sharp you want to go (if n is positive) or deep (if n is negative). But the plain use of such a system still does not give sound pleasant enough, so we are still trying to identify statistical research on the timbre and frequency complex of certain instruments so that there sounds can be emulated.

The scales we identified for use with the system based on their different cultural background and supposed emotional response for testing are

Chromatic Scale

C C# D D# E F F# G G# A A# B C

This base scale

Raag kiravani

C D D# F G G# B C

The Kirvani is a mellow night carnatic raag, that is popular in northern India as in the deccan.

Natural (pure) Minor Scale

C D D# F G G# A# C

The Natural Minor is known to be a pleasant, bright scale.

The Major Pentatonic scale (or Yaman Kalyan)

C D# F G A# C

The pentatonic scale is the most popular scale in western commercial music, it forms an important basis of Rock and Roll and Blues music. The Major Pentatonic also forms a major part of Chinese music.

The Folk Scale (pelog)

C C# D# G G# Cg

This scale was chosen as it represents music in the south East Asian countries such as Indonesia and Java. This scale is used in meditation and is said to be inherently rhythmic which would help in the notes being identified easily.

The usability testing of the system with people at BPA enabled us to optimize and tune the speech response of the system. We discontinued using the default Microsoft Speech Engine and used an Engine by AT&T which sounded similar to the one used in JAWS (a popular accessibility solution for the blind) and the new engine also had better diction. We discontinued a lot of the long responses which we had programmed and started to use shorter statements for speech feedback

Also it was found that the speech feedback should be enabled even if the user is quickly moving over an object, which we had earlier disabled. This gave them a better 'picture' of the screen they were traveling on.

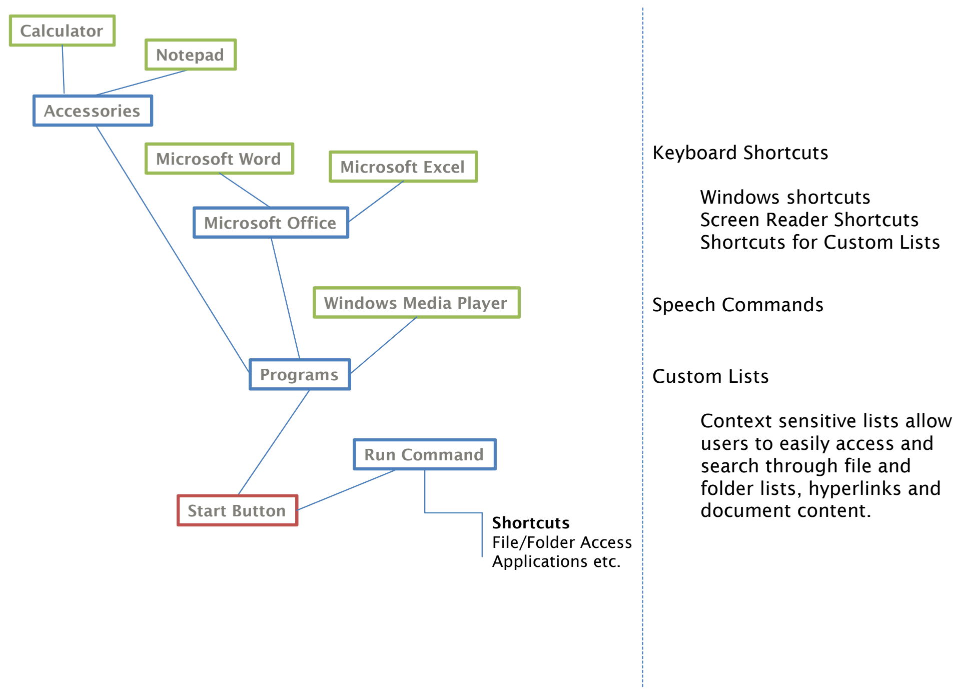
In our revision we designed the software to use both the keyboard and the mouse, which was against our previous attempts to model most tasks around the mouse. The continuous background sound of the grid was given a separate volume control so that the grid could be only a faint sound while keeping speech volume at a higher level. We also included the option to disable the continuous sound so that the grid sound is only heard on specific intervals or on a change of a set number of pixels. This meant that the user will be given a SonicGrid feedback only if he travels 10 pixels from the current position, thus cutting down on the continuous background 'noise'. Also a volume gradient was incorporated where the sound of the grid would reduce when a the speech engine was active and then rise back to the set volume after it has finished.

Further Study

The user studies carried out were primarily with people who were already well versed with using the computer using keyboard-based systems such as JAWS. These users expressed a certain frustration when trying to learn to use the mouse as they knew that they could complete the given task easily if they used the keyboard. Also it was felt that the biggest contribution of the mouse was to ease learning and initiation into computers. Ranchod bhai, a teacher at the blind school felt that the system should be tried out with young children who had not yet learnt how to use the keyboard very well. It would also ease the early days of learning the computer which involved a lot of drudgery.

The idea was taken up by our team and brainstormed upon, it was the general opinion that we should take the guidance of user interface specialists to evaluate if using our system with a new user is viable. We interacted with a couple of people at Human Factors International (HFI) regarding their views on the same. They were of the opinion that though it would allow us to test the effectiveness of the system as a new method of approaching computers by the visually impaired, we would need to have solid training material.

The concerns were that the system if not effective or if the methods for representing the new concepts were not designed in respect to the cognitive maps of the users it would make it difficult for the new learners to go back to the popular systems such as JAWS. Although it was agreed that the difficulty would not last for more than a few days, it was thought that some thought be given to understanding the mental models through which the visually impaired represent the GUI.



It construction of the mental image of a GUI is thus, different for a visually impaired, but this is more because of the construction of previous accessibility systems. Any new system should be considerate of such maps which show us that the GUI in a non-visual world can be represented spatially by connecting distinct points. These points refer to states in a task that is being performed by the user, the richer the connections between points the easier it is for a user to switch between tasks. Richer connections demand not only the presence of a short-cut or command to move from one point to another, but also grouping of states in a more semantic sense.

Hardware Learnings and Experimentation

to be filled in by shreyas

Bibliography

Blow, M. (n.d.). Unfulfilled Potential of Sensory Substitution.

Engelbart, D. *Augmenting Human Intellect: A Conceptual Framework.* Stanford Research.

Jacques Derrida, P.-A. B. *Memoirs of the Blind: The Self-Portrait and Other Ruins.*

Ree, J. *I See A Voice.*

Rowell, D. (n.d.). Auditory Display of Spatial Information.

Sacks, O. *Seeing Voices.*