November 03, 2008

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, British Columbia
V5A 1S6

Re: ENSC 440 Project Design Specifications for a Patient Comfort System

Dear Dr. Rawicz:

The attached document is a Design Specification that provides design details on how the product will actually be designed. This document will cover the system overview, system hardware, GUI software design, and testing plans to actually test the product.

We are currently working on the GUI design as it is the most critical part of our project. Also, we have ordered some third-party hardware components which will be used to execute the commands released from the GUI.

MeteorCare consist of four committed, innovative and business-focused engineering students: Aidin Mirsaeidi, Ashkan Z. Deylami, Siavash Rezaei, and Nuisha Nikkholgh. We are committed to build an innovative product that will gain a unique status in the health care industry.

If you have any concerns about our proposal or if you require more details about our product, please feel free to contact me at **(604) 312-4346** or by email at **amirsaei@sfu.ca**.

Sincerely,

Aidin Mirsaeidi

Aidin Mirsaeidi

President and CEO

MeteorCare Inc.

Enclosure: *Design Specification for Patient* Comfort System

Patient Comfort System

Team:
Aidin Mirsaeidi
Niusha Nikkholgh
Ashkan Ziabakhsh Deylami
Siavash Rezaei

Submitted to:
Dr. Andrew Rawicz
Mike Sjoerdsma
Issued date: Nov/03/2008
Revision: 1.0

Table of Contents

List of Tables

List of Figures

Acronyms

COO	Chief Operating Officer
IR	Infrared
PC	Personal Computer
UI	User Interface
I/O	Input/output
RTS	Request to Send
DTR	Data Terminal Ready
WPF	Windows Presentation Foundation
MCU	Microcontroller

Introduction

MeteorCare product (i.e., Patient Comfort System) is a state-of-the-art device intended to empower elderly and hospital patients to have control over certain activities. One of the major fears of the elderly from aging is loss of independence; losing independency is emotionally overwhelming and depressing. This device not only improves the quality of life of the patients and elderly population by giving them ability to control certain behaviours but also provides help and support to nurses and caregivers by preventing unnecessarily calls.

The Patient Comfort System is consists of a user friendly interface that runs on a computer and a hardware application. The hardware application is connected to the PC and interfaces with Meteor Care applications. The product helps patients with various activities such as controlling lights, curtains, TV as well as adjusting the bed, etc.

In the initial phase of this project, a proof of concept will be developed which will be deliverable in December 2008, followed by commercial product release.

Scope

The scope of this document is to provide design details/specification on the Patient Comfort System. In other words, this document will cover the measurements and characteristics which would produce a workable, sustainable, and pleasing product. The design specifications stated in this document are meant for the Proof of concept or the prototype. After gaining more insight in design requirements while building the product, the specifications might change.

Intended Audience

This document is intended to be used by Meteor Care design team members to build the product. The document shall be used as a guideline in developing the Patient Comfort System for the personnel of the company. Company members shall use this document to evaluate their product development progress and to keep track of their goal.

This document shall also be used as a management tool; the COO will refer to this document to ensure that the product possesses all intended functionalities.

System Overview

This section explains the general system overview of the Patient Comfort System.

Meteor care device provides the luxury of carrying out certain activities to the patients and enables them to overcome dependency problems. Bedridden patients and senior people gradually lose their independency and feel lonely resulting in depression. The Patient Comfort System helps them to regain their independency and connect.

The system composed of the Meteor Care GUI application which runs on the computer. The GUI has been developed to have simple and insightful interface so bedridden patients, older adults, and physically challenged people are able to use it without any training.

In addition to the GUI, A specialized hardware has been used so the system can integrate with the home automation hardware. The Meteor Care hardware is connected to the PC via a serial port that communicates with the Meteor Care application. The device includes a touch screen monitor along with a bed mounting pedestal to support it. Moreover, the patient’s comfort level has been carefully investigated; the product enables patients to control various devices such as the light, curtains, TV and to adjust the bed and etc. The entire embedded system diagram is shown in **Figure 1**.

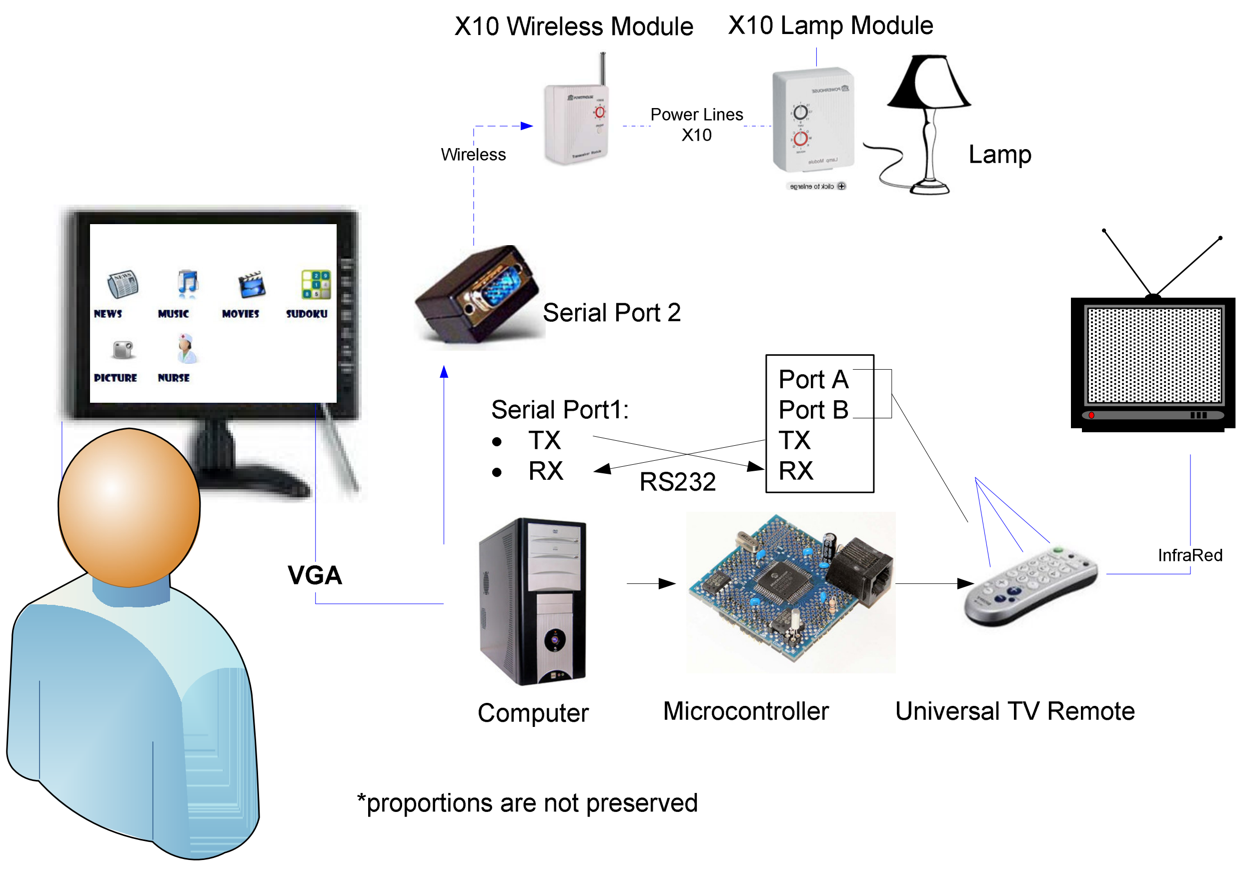


Figure 1: Entire embedded system diagram for Patient Comfort System

System Hardware

The two parts of the product design that require hardware modules are the *light dimmer* for controlling the light dimness in the room and the IR transmitter required to control the TV.

Light Dimmer System

In order to design the light dimmer module, the X10 FireCracker Kit from X10 Industries has been chosen. The FireCracker is a matchbox sized unit (i.e., transceiver) that plugs into a serial port of a PC and transmits commands wirelessly one-way to a receiving unit that's plugged into an AC outlet, which then sends the same signal through the home's AC wiring to the lamp module. An embedded system diagram is shown below that illustrates how the light dimmer system is constituted. The modules provided by X10 Industries are treated as black-boxes; the learning objective to build the light dimming system is to learn the protocol. Furthermore, the system design involves developing a software solution in .NET platform (using C# language) to send commands through the serial port to the transceiver.

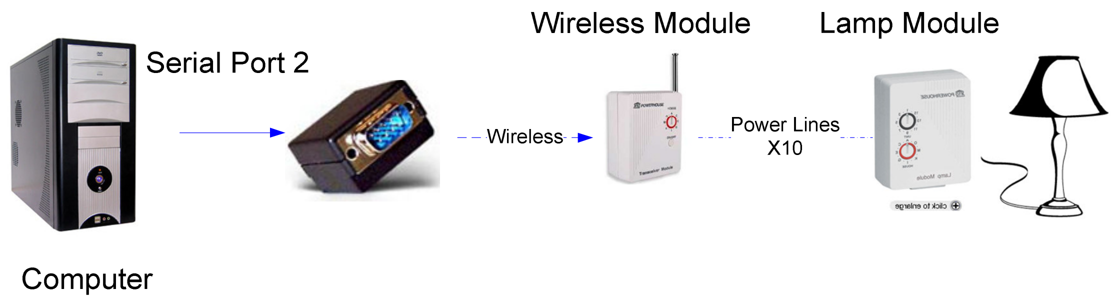


Figure 2: Embedded system diagram for light dimmer module

The lamp module is recognized with a house code, and a unit code. The lamp module/controller can be set to a particular house and unit code by rotary switches. Since this product will be designated for one room and one lamp, then the house and unit code will be fixed in the commands sent. However, there can be more lamp controllers plugged into the AC line which would have different unit numbers but the same house number. There are several different commands that can be sent out: on, off, dim, brighten, all lights on, all lights off and all controllers off. The on and off signals are sent to a house and unit code. This causes that controller to go into a listen mode after which it will respond to dim and brighten commands. Each dim or brighten command causes about a 5% difference in the dimness or brightness respectively. The all lights on and all lights off are targeted to only a house code which would cause all light controllers (up to the max of 16) with that house code to respond appropriately. Finally, the all units off will turn off all light modules off on the designated house code.

The Firecracker does not rely on normal serial communication; instead it is driven by the RTS & DTR signal serial lines which also provide its power. The command frame is pretty simple, consisting of only 5 bytes. **Table 1:** Message Format transmitted to the receiver outlines the message layout. A couple of tricky areas are initialization of the device and clocking out the signals to the RTS/DTR lines with appropriate delays between bits. Initializing the device is achieved by setting the RTS/DTR lines low for a short time and then bringing them both high with another delay before commencing a message.

Table 1: Message Format transmitted to the receiver (iii)

Header	House code	Unit code, Function/command	Footer
0xD5,0xAA / 11010101,10101010	01100000 <u>or</u> 01100100	Refer to tables 3 and 4	0xAD / 10101101

Sending out bits consists of starting out with both lines high and setting DTR low for an on bit and setting RTS low for an off bit. In both cases, there must be a delay of at least 500 microseconds before setting that same line high again. This process continues until the entire 5 bytes are sent out. The house code that we will use will be “01100000” for units 1-8 and “01100100” for units 9-16. **Table 2:** Unit codes and ON/OFF functions lists the byte to send to turn a controller either on or off. Note that controllers 9-16 have the same code as 1-8.

Table 2: Unit codes and ON/OFF functions (iii)

Unit #	ON		OFF	
	Hex	Binary	Hex	Binary
1	0x00	00000000	0x02	00000010
2	0x10	00010000	0x12	00010010
3	0x08	00001000	0x0A	00001010
4	0x18	00011000	0x1A	00011010
5	0x40	01000000	0x42	01000010
6	0x50	01010000	0x52	01010010
7	0x48	01001000	0x4A	01001010
8	0x58	01011000	0x5A	01011010
9	0x00	00000000	0x02	00000010
10	0x10	00010000	0x12	00010010
11	0x08	00001000	0x0A	00001010
12	0x18	00011000	0x1A	00011010
13	0x40	01000000	0x42	01000010
14	0x50	01010000	0x52	01010010
15	0x48	01001000	0x4A	01001010
16	0x58	01011000	0x5A	01011010

Finally, **Table 3** lists the commands that could be sent to the lamp module for turning on/off or adjusting the dimness/brightness of the lamp.

Table 3: Command names and corresponding codes (iii)

Command	Hex	Binary
Dim	0x98	10011000
Brighten	0x88	10001000
All light on	0x90	10010000
All light off	0xA0	10100000
All units off	0x80	10000000

IR System

The IR system is solely used to control the TV in the patient's room. The hardware components used in this system are a universal remote control, and a MC9s08AW60 microcontroller. In order to make the implementation of the IR system easier, a universal remote control is going to be used for IR signal transmission instead of building an IR transmitter from scratch. An embedded system diagram that illustrates how the system components are connected is shown below.

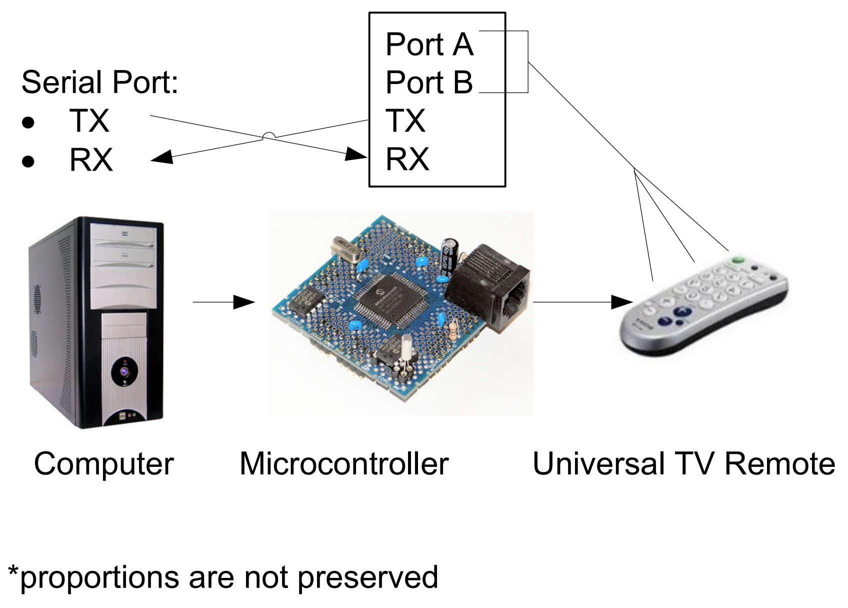


Figure 3: Embedded system diagram for IR system

MC9S08AW60 microcontroller

The microcontroller is connected to a PC via the RS-232 serial port to receive commands from the .NET platform, the platform used to design the software portion of the product.

Since the MC9S08AW60 microcontroller has 54 general purpose I/O pins, the internal circuitry of each button of the remote control is going to be connected to an output pin of the microcontroller. For example, if the channel 1 command is received by the microcontroller, then the corresponding output pin connected to the channel 1 circuitry of the remote control would be set high for appropriate IR signal transmission. **Table 4:** Mapping table for microcontroller output pin and remote control button connections illustrates the connections between the microcontroller output pins and the buttons of the remote control. **Table 4:** Mapping table for microcontroller output pin and remote control button connections is only used to help the reader understand the detailed design specification; hence there could be more output pin-to-button connections if more TV control is required.

Table 4: Mapping table for microcontroller output pin and remote control button connections

Button	**Microcontroller Output Pin**
ON	PTA0

OFF	PTA1
0	PTA2
1	PTA3
2	PTA4
3	PTA5
4	PTA6
5	PTA7
6	PTB0
7	PTB1
8	PTB2
9	PTB3
volume down	PTB4
volume up	PTB5
channel down	PTB6
channel up	PTB7

For sending commands from the PC to the microcontroller, a dispatcher unit has been developed that acts as a serial port interface between the microcontroller and the PC.

Protocol needs to be formulated so that any valid command sent to the microcontroller could be decoded and the corresponding output pin of the microcontroller would be set high which in turn would trigger the appropriate action on the remote control for IR signal transmission. These protocols, and the way of implementation is described in proceeding section titled Microcontroller Communication.

Universal Remote Control

As mentioned before, a universal remote control is being used in the IR system design as to make the implementation easier. This way, the IR transmitter IC is ready for use and only the circuitry paths of the different buttons on the remote need to studies so that the output pins of the microcontroller are connected to the right junctions on the remote control circuitry. This will be accomplished by purchasing and opening a universal remote control and by using a volt meter, determine where the right junction is for each button so by sending a high voltage signal to that junction, the corresponding IR signal would be transmitted to the TV.

Microcontroller Communication

The low level functions in the microcontroller would implement the hardware level interactions required by our project. In order for us to reach and call these functions from the GUI which runs on the main PC, we need to implement a message handling system.

We call this message handling system the "Dispatcher". Dispatcher is basically a protocol layer over the RS232 communication. Here is a simple visualization for the MCU part of the dispatcher:

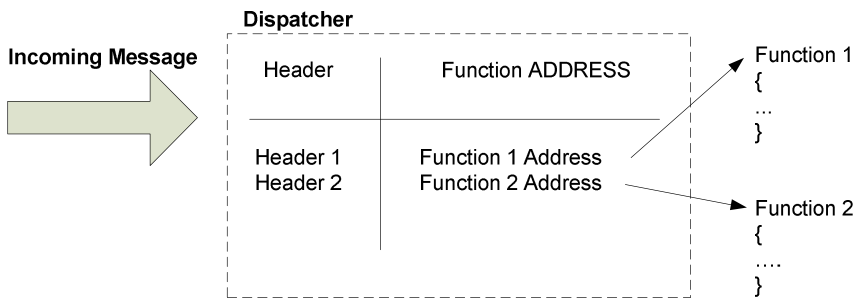


Figure 4 Simple Mental Image of How Dispatcher work

Quick Setup

A function in the microcontroller registers itself as the handler of specific header. For example, we have a function that is called THEFUNC, and it should handle the messages with header of "LIGHTS".
We register the function as:
dispatcher_addFunction("LIGHTS", &THEFUNC);
From now on, every incoming message with the Header of LIGHTS would make dispatcher call "THEFUNC", and pass on all the consequent arguments to that function.

The function of THEFUNC can be implemented as simple as:

```
    void THEFUNC(byte * structure) {
```

```
    temp_cheragh = structure[0];
    PWM1_SetRatio8(temp_cheragh);               //Set the pulse width modulation Ratio

  }
```

Protocol Definition

On the low level the dispatcher's protocol is pretty simple, it is:

/,[FUNCTION HEADER]**,**[FUNCTION ARGUMENTS]**,/**

There are only two things in there:

1. Function Header: Represents the header of the following arguments. Dispatcher decides which function to call based on the registered Headers in its table.
2. Function Argument, can be any number of bytes. Ideally, we create a structure which is a collection of those bytes, and then the function that handles the dispatched message would treat the received bytes as one structure.

Obviously the functions in the GUI level don't need to take care of this protocol. A simple Dispatcher function on the GUI side, would convert the given structure to match the protocol then send it to the microcontroller through RS232.

GUI Software

Introduction

Most dominant feature of a product such as ours should be its ease of use and interface. Therefore a lot of thought and consideration has gone into designing the user interface. The GUI does not necessarily have to pack a lot of features and functionalities; however, it should be very good at what it does. To achieve our goal we had to give the user clear visual feedback for his/her actions. Since we perceive the world in 3D, the decision was made to organize the menus in 3D in a smart manner. This helps the user to easily see the relation between menus and icons. Moreover, this method delivers very useful information yet keeps the screen visually clear and uncluttered.

Architectural Strategies

C# and XAML are selected as the languages of choice for developing the GUI and the business logic behind it. Since ease of use and great user interface is the primary feature of our product, WPF is employed to enhance the graphical tools available to the programmer to develop the GUI. In a nutshell, WPF is a graphical subsystem of the .NET frame work 3.0 and 3.5, which uses a mark-up language, known as XAML for rich user interface development (i) (ii). After choosing WPF, the choice of C# as the programming language is the next logical step because of its integratability with XAML code.

Interaction with GUI

The main menu of the GUI is depicted in Figure 5.

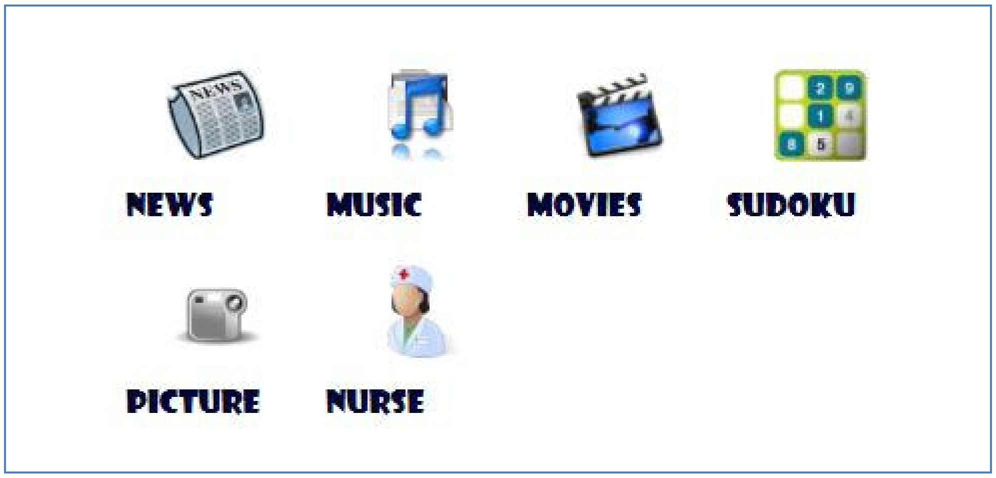


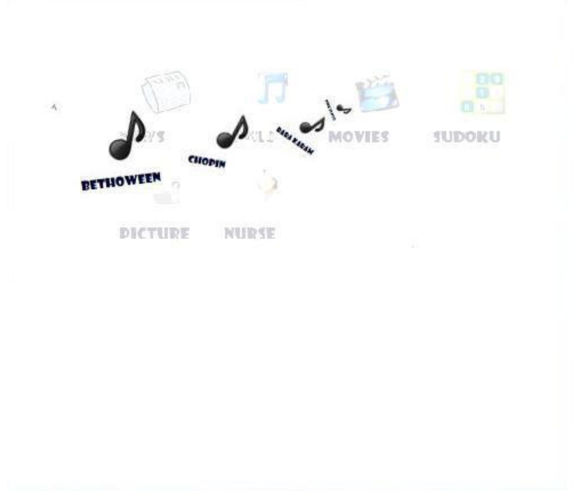
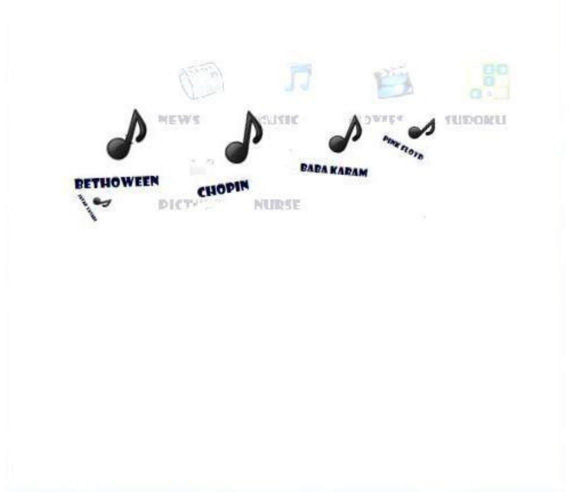
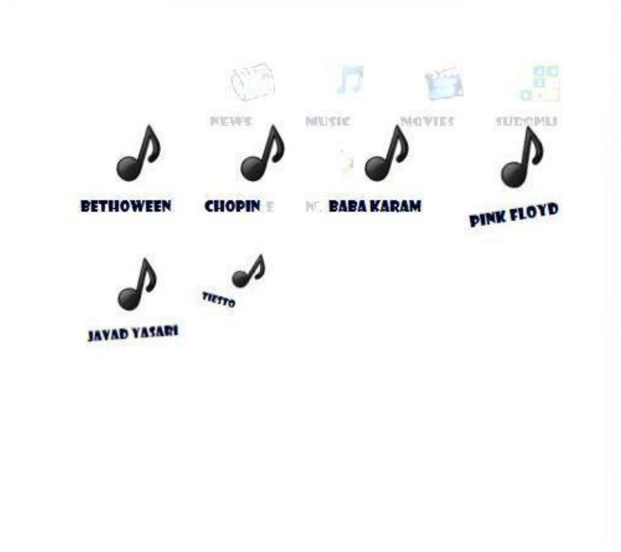
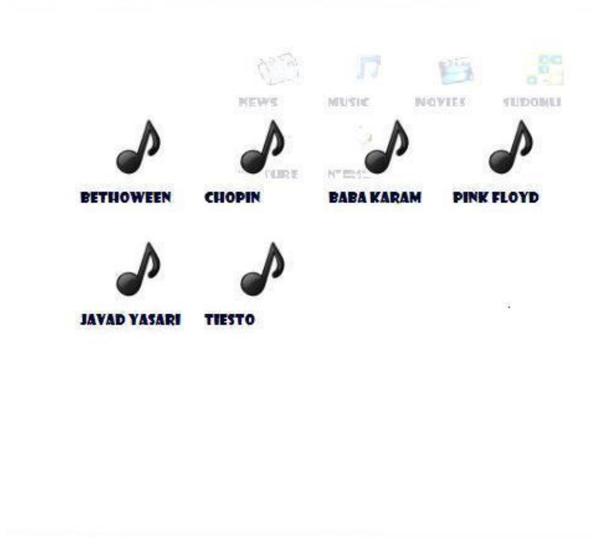
Figure 5: Main Menu

User can trigger the icons by touching them. In the case of patients without the ability to use touch screen, the GUI can be configured so the icons enlarge as the patient tabs through them to give visual feedback to see which item is currently focused.

The following goes to describe different scenarios when the patient selects each of these icons on the main menu.

Music: When Music icon is pressed an animation is played which displays Artist icons jumping out of the Music icon and settling in the screen. At the same time, main menu icons shrink and their opacity decreases. That gives the feeling that the main menu icons are pushed back in space. So the user can focus on the icons that are on the top. The table of figures below is sequence of snapshots taken from the animation.

Table 5: Animation snapshots

 BETHOWEEN CHOPIN BABA KARAM MOVIES SUDOKU PICTURE NURSE Shot 1	 NEWS SUDOKU BETHOWEEN CHOPIN BABA KARAM PINK FLOYD NURSE Shot 2
 NEWS MUSIC MOVIES SUDOKU BETHOWEEN CHOPIN BABA KARAM PINK FLOYD JAVAD YASARI TIESTO Shot 3	 NEWS MUSIC MOVIES SUDOKU BETHOWEEN CHOPIN BABA KARAM PINK FLOYD JAVAD YASARI TIESTO Shot 4

When the artist icons are pressed, the songs pop up in the same manner that the artists appeared, and the artist's icons are pushed back in the same manner the main menu was pushed back. When a song icon is clicked the song starts to play and the appropriate controls (i.e. Play, Stop, Next Song, Previous Song, and Volume) slide up from the bottom of the screen.

News: When News icon is pressed an animation is played which displays news headlines jumping out of the News icon and settling in the screen. After reading the headline, if the user is interested to know more about the subject, they can touch the headline and a new window will pop out that contains the detailed story of the headline.

Movies: When Movies icon is pressed the movie icons (each representing a movie) will pop out of the Movies icon. Pressing the movie icons results in another window popping out which shows the movie and contains the appropriate controls such as Play, Pause, Stop, Fast Forward, Reverse, and Volume.

Sudoku: When Sudoku icon is pressed a window pops up from the Sudoku icon which displays the Sudoku grid with appropriate control.

Picture: When Picture icon is pressed, Album icons pop out of the picture icon. When each Album icon is pressed picture thumbnails pop out of the Album icon. User can browse through the thumbnails by touching the screen and sliding them past. If the user touches a thumbnail, it will enlarge so he/she can see the picture in greater detail.

Nurse: When Nurse icon is pressed, it gives the user a visual feedback to let him/her know that a call is being made. Behind the scene it sends a message to the appropriate X10 module that alerts the nurse.

In all the cases explained above when an icon is touched that triggers another menu or window to open, the parent menu is pushed back in the space (as illustrated in Table 5: Animation snapshots). Furthermore, to go back to the parent window, the user could touch anywhere on the screen (except on the icons of course). Doing so triggers an animation showing the child icons to jump back into their parent and the parent menu is brought forward in space.

Test Phases and Cycles

When it comes to luxury products or products that their audience are technically challenged, a simple product error would mean that user would abandon using the product. The main issue arises from the fact that user is unable to determine whether the problem is initiated by him/her or from the software.

Consequently, a considerable amount of thought and ingenuity should go toward the test processes to make sure the product is in fact close to bug free.

Here we state two general types of tests that seem mandatory for our product: Project Integration Test, and Operation Acceptance Test. In the proceeding text, we discuss each type of test accompanied by their detailed algorithms and schedules.

Project Integration Test

When the project parts are assembled together, a series of tests should be done to ensure that they function correctly together. Here we present some general roots of errors in the project integration test:

1. Two components that normally function well independently, malfunction together due to miscommunication
2. Two stable components placed in a feedback loop function in an oscillatory or unstable manner
3. A component is dependent on another component which is faulty

Here is our test plan in order to identify each of those integration errors:

1. We write a software code that generates a table of all possible communication commands for the components that talk to each other. The test code would loop through the table in a random fashion, to test each command. The results of the test would then be written in a log file categorized by satisfactory and non-satisfactory.
2. In our case there is no feedback system, as there are no sensors implemented anywhere in this current version of the product.
3. In order to have a robust system, the protection against wrong inputs from other faulty components should be implemented in the component level.
   Afterwards in the system level, we would then generate a table similar to the table mentioned in point #1, with faulty data. If a part of the program crashes, then the component responsible should be fixed in a component level. This test should cover the errors which arise from Faulty Components or missing components or faulty communication lines.

Operation Acceptance Test

Software Test

For details on what the components are for the software please refer to

GUI Software section. In this section we discuss the parts of the operation acceptance test for the software. We note here that our software is created by integration of classes that each takes care of one functionality of the program. If the program is to work flawlessly, each component should work seamlessly with others.
This in return means that all the rules of Object Oriented Programming should be followed, and we need to ensure that each function can handle wrong inputs in a descent manner.

The test algorithms written in this part is generally simple and very straightforward. You call the functions of the program with null referenced objects and see if they break down or not. In the case they breakdown, a record should be written down with details of the inputs and the problem. This way we can address the problems in a systematic way.

This approach both applies to the code written with C# for the GUI, and C++ functions written in CodeWarrior environment for the embedded system.

Navigations

The user should be able to go through the menus and reach his/her objectives with no problem. This test is a test for human interaction simplicity rather than a software challenge or design.

In order to test this part of the program, we need to create simple case studies with specific objectives, and ask a participant to do the specific required task. The participant should be chosen carefully to represent the target market and audience.

If a desired task is carried out with no problem, then it means the designer has done a decent job. If the task is accomplished with a certain level of difficulty, then the designer needs to go back and redesign the process.

Through this re-design cycle, our product can achieve a better usability.

Hardware Test

The hardware test generally should contain couple of aspects such as:

1. durability
2. functionality
3. error handling margins

In our case, as we are not mass producing the product, we don’t need to worry about the hardware test systems implemented automatically.

Also, the general trend of technology shows most technological products such as cameras and TVs go out of style much sooner than they start malfunctioning. Consequently, here we only need to worry about the functionality of the hardware components.

Most of our hardware components are black boxes, bought from reputable manufacturers. Those products obviously have gone through cumbersome tests before being on the shelves of the stores. Consequently, we won’t be testing them for their product specifications. Here we would take the manufacturer’s word for what the product is and what it does.

However, for the components made by our group we need to test for:

1. Soldering
2. Connections
3. Wiring
4. Ground protections if the powered by high power voltages
5. Dropping them. Running over them. Putting them through the spin cycle.

References

i. http://en.wikipedia.org/wiki/Windows_Presentation_Foundation
ii. http://msdn.microsoft.com/en-us/netframework/aa663326.aspx
iii. http://www.codeproject.com/KB/library/x10Demo.aspx