M-Board – The Mind Keyboard

June 8

2012

ICT Engineering program – VIA University College

Author: Claudiu Lucian Manea, 130935
Michal Slavkovsky, 132630
Thomas Rheder Christensen, 126589

Supervisor: Allan Henriksen

Project Report

Table of Content

Contents

List of Figures

List of tables

Abbreviations

MBoard – The name of the application

XML – Extensible Markup Language

RSS – Really Simple Syndication

GUI – Grafical User Interface

intPtr – Integer Pointer

STAthread – Standard WPF GUI thread

MVVM – Model – View - ViewModel

TDD – Test Driven Development

WPF – Windows Presentation Foundation

Dll – Dynamic link library

Tx – transmit pin

Rts – ready to send pin

Dtr – Data Terminal Equipment

Abstract

We have made a system to help the disabled people, more accurately people that are paralyzed from the neck down. The system is called Mboard but can do so much more than that. The system consists of several different parts where the most important of course is the keyboard part, but we have also made a home automation module that can be used to control the environment around the user. When you're caught in a bed you can get a bit bored so we also have a game ready to play, even thou it's a simple card game called High-Low. The last feature we decided on was the news so that the user has the ability follow the events happening in the world. To get back to the Writing part we made a keyboard that could be used only with your mind, this in itself seems incredible but we did a bit more and implemented a text to speech function as well and to speed up the writing we moved the keys around a bit so you would get to the often used keys faster.

1. Introduction

1.1. Background Description

Although nowadays some would say that we are surrounded by technology like smart phones, TV's, and computers, and that this allows the users to communicate easily. But there are still certain people/groups that don't have the opportunity to use and control them as we can. Physically impaired people cannot move any body part at all, but there brain is fully functional. Therefore, we would like to develop the "M-board" in cooperation with Horsens Kommune that will engage those people with their environment more seamlessly and improve their lives, bring them closer to society.

The primary focus of our project is to build a digital spelling table (M-board) with a cursor hovering over the letters and along with the Emotiv EPOC device we will be able to detect specific impulses when the subject would want to chose a letter. This way they will be able to formulate words and even sentences, leading to their ability to communicate with others. The fact that the product requested from us is quite basic that doesn't mean we lack new ideas in which we would make it even better.

We think this is an important area to work in since we today have the technology to make the everyday life easier for a lot of people that are disabled in different ways. The reason for us to focus on this specific group is that the technology is cutting edge and we think it has the potential to become very big.

Even though today there already exists a lot of applications and instruments that help those people, for example: controlling PC using the eye or iTongue control that is using a tongue for control various devices around. We would like to go more forward and to create the application that allows the user a control with only a thought. Just wonder, when the user will be able to control the house or other environment with his mind.

Seeing that this is a new technology it has a lot of room to evolve and the possibilities are endless. After we develop the basic components, we can build an endless number of software solutions.

The basic concept behind is to use the Emotive EPOC headset that can detect and process real time brain waves. It measures a wide range of thoughts and emotions and allows using brain activity in the application. The human brain is made up of billions of neurons. When we do various activates, neurons transmit messages with each other, and we can measure those signals with electrodes and amplify them and then later use them in our application, to the various actions we want to perform.

The Emotiv headset uses 16 sensors that measure the electric signals in our brain. They get readings in pairs and the differences in readings are then treated as the signal. When the application will ask the certain action, it will compare how the values from the electrodes change. When the change will be similar, the application accomplishes the calling action. The Emotiv headset is able to recognize expressions, actions which transmit over Bluetooth and the Bluetooth dongle to the application that we can read. So, the Emotiv headset is kind of the interface between the user and computer to handle the user's actions.

Because the brain waves are relatively different for every user, we need to calibrate the application before use. Then we can save these parameters for every user, and use them for controlling our "M-board". Every user will have his own account with his own record data, so it can be applicable for many users.

As we mentioned at the beginning, we will mainly focus on physically impaired people, those who can't communicate or express their needs. Therefore, the Mboard will be the main part of the program, which allows writing, reading news or doing a small work on the PC.

Although our software will not have been using all of our ideas, however it will help and make life easier for many users. This is a great challenge for us to create the program for the people need to depend on others. On the other hand working with the Emotiv Headset will give us the opportunity to learn the latest technologies, that we think will be used more and more in the near future, in normal everyday life and in the gaming industry.

1.2. Problem Formulation

As we went through the problem, we figured out, that we have basically two main parts of the application which we have to handle. Firstly we need to use the Emotiv Software Developer's Kit to define a bridge between the Emotiv headset signals and our application to be able to control various features in there. The second important part is to use suite sequences and implemented them with required functions.

1.2.1. Connection:

How to connect our application to the Emotiv Engine? How to get the source of all the signals in our application? Those questions are probably essential before we will be able to continue forward. We need to connect and build our own bridge that will connect the Emotiv device and our application. We also have to decide whether we are going to handle user profile in our application, or we will use the Emotive Control Panel. Maybe it will be better to have our own profile and just call user profiles from the Emotiv engine.

1.2.2. Structure of the events:

How to handle various events from the headset for more users? Because the headset generates various events (four categories) we need to handle those events and it might be good to create the data structure, where we can find out what event was generated according to the user's ID.

1.2.3. Selecting suite Commands:

We need to ask ourselves, what kind of functions we want to control? Which means that we have to consider how to control the navigation panel and the M-board? For example go down, up and enter functions. So we need to decide what sequences (signals) from the users would trigger those functions. The Emotiv software provides us with 4 main categories of data: Expressive, Affective, Cognitive and Gyroscope data.

We need a control set that would work for people with a minimum amount of training. We made the decision to use only one cognitive function from the headset. This allows user with much less training to use the application on full capacity.

1.2.4. Designing the Graphical User Interface:

How to design the GUI (navigation panel)? How to make the design of the keyboard and organize letters on the keyboard?

The GUI design must be as simple as possible to be easy to navigate through both when it comes to the keyboard and also through the navigation panel. The user should have feeling that automatic writing is done with the keyboard and computer. We also need to consider, how to design efficient mapping of the keys on the keyboard. We shouldn't use the common keyboard design, because the writing will be too slow for our application. There should be done a new algorithm.

Other questions: What size of the font we should use? How to change the type of keyboard from English to Danish and back?

Obviously, some of the people need to adjust size of the font when they are writing and also change settings such as the keyboard language, therefore we should consider it in our design, the suite place for the writing settings and easily manipulate with them.

1.2.5. Store Data:

How to save data? Should we store the data in xml files or a Database? Do we need to save any data?

That is a good question. Maybe we don't need to store any data, but we have to consider the best solution. We can call the information about the user profile, configurations and signals from the emotive control panel or build our own storage.

When we decided to build our own storage, then we have to consider the amount of data that we will be storing. Probably, there will be very small amount of data, therefore we would store data in an XML document and re-write the file any time any data changed. So it would be useful for configuration files and also for the user's profiles. We are expecting that the XML files never grew larger than 100kb.

2. The Mboard System

2.1. Architecture

Like every good house, software must be built upon a solid foundation [4], otherwise we will be face many challenges such as unstable application, difficult to support existing or future requirements and at the end the difficulty to deploy and manage such system. Therefore we should be able to design and implement the software, which is simple and powerful. The code should be clearly written, which is easily to maintain and modify. The object-oriented design and clean implementation with the consideration for the end-user and final goal are giving good reliability for the project. We also have been focusing on these major elements that we had used in the application. We split application to the three major parts so that EmoEngine tier, Domain Model tier and Presentation tier. Our architecture can be seen in Figure 1.

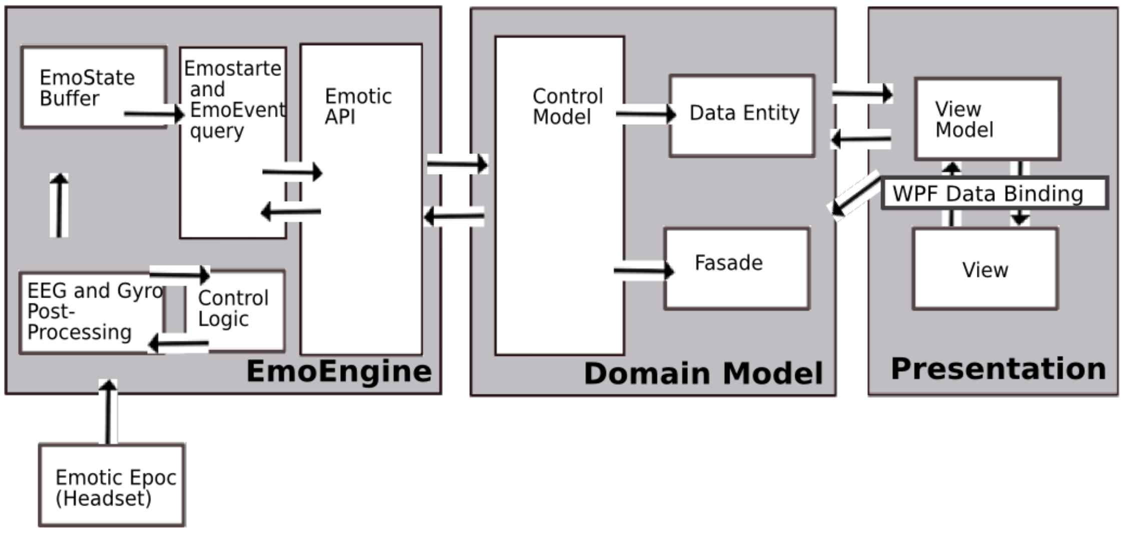


Figure 1 Application Architecture

The Emotiv EmoEngine refers to the logical abstraction of the functionality that Emotiv provides in edk.dll [3]. The edk.dll is a collection of small programs, which can be called when we need to use them in our application that is running in the computer. We can say that edk.dll is supporting operations and communicate with the Emotiv headset as device driver.

We can see in figure 1 that EmoEngine is receiving brainwaves (EEG) and gyroscope data from the headset, manage user specific settings, performs post processing and translates the Emotiv Detection results into an EmotiveState. Then we can modify or retrieve these EmoState setting by Emotiv API functions in our application. So An EmoState is current state of users' cognitive state that will be control functions in the application. More detail description about an EmoEngine, how is implemented in our code will be presented in the next section.

As we explained at the beginning, the next tier we plan to add to the existing architecture is a domain model. The Domain Model is entailed listing the entities and defining relationship between them. Or we can say it has contained objects that include data and behaviors that describe basic architecture application. We added components, objects to it and the classes responsible for handling collections of entities. These classes define the structure and behavior of entities, characterize the outputs of these domain; how the result from these domains are carried out by a set of functions. Then these outputs provided by functions are retrieved for the presenter tier and EmoEngine tier.

A presentation tier is actually interfaced between the user and application. It will be responsible for the communication with the user. As presentation system we decided to use Windows Presentation Foundation (WPF) because it has strong ability to develop an application using both markup as code-behind and this is a big advantage for us. Generally we have using Extensive Application Markup Language (XAML) to implement the appearance of an application while managing its behavior is done by code behind.[5]

So we can simultaneously working with the design and the code. Maintenance and development is giving us more flexibility and options. With WPF the bindings makes it easier for updating data from the EmoEngine. Each ViewModel parameter is set as the data Context of its pair view. When the property value in the view model is changed, then this change automatically propagates to the View. When the user make action on the view object, then a command execute on the View Model to perform the action. The Model view performs all modifications made to the Domain Model.

Implementing a three-tiered architecture for our application will help to achieve the final result. Also the architecture gives us excellent level of flexibility and extensibility in the long run. To divide the parts of the application to smaller components encourages reusability. When we decide to make changes in the design of the application or add new features then we don’t need to change much of what already exists. Also smaller part of code is easier to analyze.

Also a good thing with our architecture is that, when the domain one of tier change, the other tiers usually remain almost unaffected.

2.1.1. Analysis and Design

The first step in good analysis is figuring out potential problem. Sometimes the applications tend to be much larger and have complex structure and therefore it can be more difficult to develop them. But the importance of application is to understand end user’s needs, analysis the problem to the detail and then we can design the architecture in the iteration (life cycle) process.

By the object-oriented analysis we have been tried to identify the type of objects that map into elements of the domain application. This process has helped us to find the relationships between the different types of objects considered as class instances. The class is describing the object behavior that’s provided their constraints and relationship to other classes. We had identified all class information that considers public or private access specified.

We have used the standard methods like Unified Modeling Language and Fundamental modeling concepts during the analysis that defines all of these specifications up front. We started with the domain analysis that gives us a conceptual framework of the objects in our architecture. Then we defined functionality of the application with the use case and defined design of the application with following techniques such as the activity diagrams, sequential diagrams and class diagrams. All this process has been iterating and changing continually in the process development, therefore we are showing mostly only final iteration of the development process.

2.1.2. Domain Model

The principle of the domain analysis is seeking to identify the classes, operations, relationships and objects that are common with the given domain. This helps as to understand, to think and focus on the basic concept of the architecture. It will help us to work with the use case and workflow modeling.

In general there are steps to establishing Domain Model. First, we have defined object, then identify objects and at last to establish a model from this objects.

2.1.3. Define Object

1. The M-board will be an application with WPF user interface, but it must have a sufficiently architecture so alternative functions may be developed
2. The M-board will allow a user to write text, with the Emotiv headset

2.1 The user must be able to delete written letters.

3. The user must be able to change the size of font.
4. The user must be able to change the keyboard language.
5. The m-board must remember the changes in the setting by user.

5.1 The setting must be defined in XML

6. The m-board shall be used to read the news

7. An assistant user must be able to create a profile for the user, so that the system remembers the user's name

7.1. The system shall maintain a list of profiles in XML files

7.2. When a user log in, the system must load his or her recent settings

7.3 The user doesn't need to insert a password

8. The user must be able to control (home automation)

9. The user will be able to play the simple game

9.1 It would be nice to remember the statistics

10. The user must be able to take a training to control movement by the Emotiv headset, with the help of the assistant user

11. The system must remember the parameters from the last training

12. The system should show the status of the battery.

13. The system should show the connectivity with the headset

14. The system should show a progress bar of the signal

15. Each function must be controlled by the Emotiv headset in the application

16. The m-board will be capable of serving only one user in time.

2.1.4. Identified Objects

Now, we have marked the red nouns that could be key words for our domain diagram. We have taken them and make a list of candidate domain classes. If we have found duplicates then we eliminated them from the list. Also we eliminate keys that could be used as parameters in the domain classes.

Headset	Game statistics	List of setting
Font Size	Training	User
Language	Home automation	Game
Settings	List of users	XML file

The next step is to draw these classes in a UML class diagram and identify any associations that are necessary. In this iteration we will not identify attributes. The result domain diagram can be seen in on below. When we compare the domain diagram with the final class diagram, we can see that are many additional object that were needed to define later. Our domain diagram is not including the presentation tier and its classes.

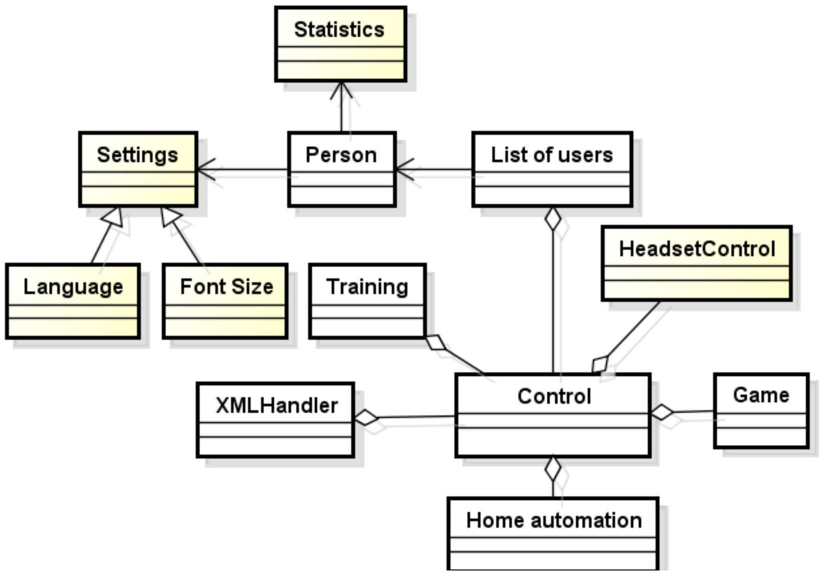


Figure 2 Domain Model

2.1.5. Use Case Model

Use case diagram better describes the possible sequences of interactions between its external actors and the system. [1] We started building our use case diagram in the context of the domain model. As we mentioned in the previous chapter, a domain model captures the most important types of objects and events in the context our case but the use case to show all of its available functionality. In general, a use case gathers requirements of a system from the customer, which is a Horsens Kommune (or supervisor). We were discussing the functional requirements with our supervisor; he was pretending to be our customer and we defined together the main purpose of the system, than analyzed the functionalities in the use cases that had to be done before first release.

So we can say that the use cases are nothing but the system functionalities written in an organized manner. [2] We can better distinguish the requirement of the system including internal and external actors. In our case the actors are persons that will play a particular role when they will use our system. We defined two main actors of a use case, a user and an assistant of the user. The main different between them is that the an assistant user is a person that can be able to control profile for the user on the system, booting an application, prepare headset for the user, generally speaking assistant user will just help a user with the application.

A user is an actor that will be able to control application only with the headset. So he is not able to pressing buttons by a keyboard or a mouse. The List of actor can be seen in the Table 1.

Name	Characterization
Assistant User	The person able to run an application, prepare a headset for the user and create an account for the user.
User	The user working with this application continuously. He will use only a headset that will help to control the functionalities of the application.

Table 1 Actor List

As we have already said, a use case diagram represents a task – use case performed by actors the use case diagram can be seen in appendix C. We can also see all use cases for our application. In addition to showing which actor perform which use case, use case diagram can show relationships among use cases as well. The first relationship is included between logout and close application use case. When the user assistant closes the application at which the use case generating logout is included. This means that at this point the entire systems will logout the user and close.

The second relationship is extending. This means that one use case may be used to extend the behavior of another. That can be seen between the create user profile use case and login use case or training user and cancel training use case. Every use case is described on below, for this we used a template, which is easy to read. The use case descriptions can be seen in the appendix C.

2.2. Class Diagram

The final design of the class diagram can be found in appendix D. the trip from the initial design to the one we have now has been ever changing all the way through, we made research along the way and as we were adding new objects and subsections then we got a design that showed all the relationships between objects. This will be described in much more detail in the coming sections.

2.3. GUI Skeleton

2.3.1. Analysis and Design

The Graphical user interfaces (GUI) has high importance in our process design. We don't mean a visual design, but a functionality design of the interfaces. Especially when we have two types of the same actors in our application. The assistant user will be able to control application with the mouse or keyboard, but the end-user will be able to use only headset that should invoke the button events. For this reason we had to constitute a design that will be considering the end-user needs.

As we mentioned at the beginning, we would like to use the WPF that easily allows creating, displaying and manipulating user-interfaces. We want to have simple design that will be only contain one type of controls. We have to avoid the problems that might occur when we will mix the various type controls such as combo boxes, radio buttons or others that we are hardly manipulate by headset.
Therefore we decided that we will be only using buttons in the application.

The main problem that we was the most challenging is that how the user will be control buttons and navigates between panels with the one cognitive state. The application should anticipate operations between panels. We decided that timers should anticipate navigation through the controls (buttons) on the panel and the user must make decision in a given time interval by the headset if he wants to invoke buttons event or not. If not then we move focus on the next buttons and then again will have some time to invoke the button.
This way it could work but we have still too many buttons on the panel so it will take a long time when we will flow through all buttons. We need to improve this technique and reduce buttons.

Therefore, we changed our design again, and decided to split the functionalities between the six panels and this way we reduced the amount of buttons on the one panel. This gives the user less time waiting for the buttons. Furthermore, we split every panel between sections the header, body and footer and we want to control header and body buttons separately. So when we load panel, body timer we will start the flow through the button in the body section, when user will decide go to the next page, then he must select the button Back to Menu, and the panel start flow through the header buttons. This way we are able to control panels and also functionality in the application by the headset.

The design is based on a multipage application in WPF. So we have one window but the multiple pages. Technically we will switch between panels. As we set the panel will be split to the header, body and footer section. The windows with the multipage are shown in figure on below. The main idea this design is when you press Button1 then we will navigate to the page 1 and so on. The main window will load a new page with the same header and footer section. The previous page is disposed.

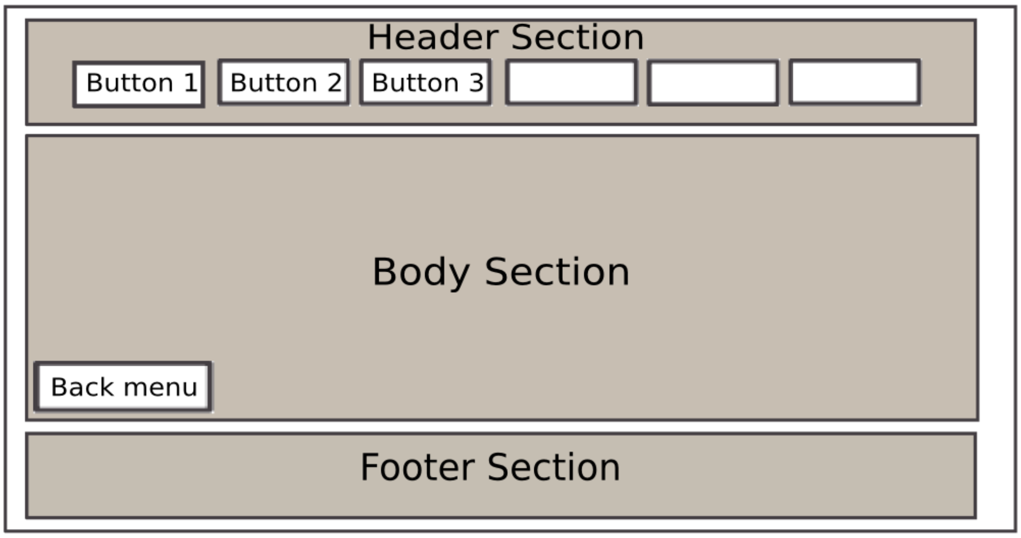


Figure 3 GUI Structure of the application

2.3.2. Implementation

First, we had implemented two navigate function in the core of main window class. The functions are just navigated to the new page and takes parameter UserControl.

```
public void Navigate(UserControl nextPage)
        {
            this.Content = nextPage;
        }
```

Then we implement interface ISwitchable, that contain one method UtilizeState and passing object.

```
interface ISwitchable
    {
        void UtilizeState(object state);
    }
```

We created a new class Switcher, which will contain two methods Switche from the main window. But we are showing only one because we haven't used the second one so far. The static method Switche is passing a new instance of User Control and calls on the newly instantiated main window. When we call navigate then we pass in a new instance of page and the content of the main window class is filled with the user control.

```
public static class Switcher
    {
        public static MainWindow pageSwitch;

        public static void Switche(UserControl newPage)
    {
        pageSwitch.Navigate(newPage);
    }
```

As we said we have six user control(Pages), then we need to switch pages between them. When we click on the button from the menu, the vent handler for the button is called and then we call the static method on the Switcher class, passing a new page. Example code on below is showing event handler that is calling News page panel.

```
private void Button_Click_5(object sender, RoutedEventArgs e)
        {
            Switcher.Switche(new News());
        }
```

2.3.3. Test

For this test we did some basic interface testing to see if we could navigate the buttons between panels.

Description

This test is going to make sure that all the panels are loaded when the header button is pressed.

Scenario

- Start the program
- Navigating through the menu buttons

Preconditions

The main windows is loaded and running without problems.

Element	Description	Expected Result	Actual Result
User Panel button	The user panel is loaded when the button n is pressed	The panel has been loaded on the main window.	The panel was loaded
Option Button	The option panel is loaded when the button n is pressed	The panel has been loaded on the main window.	The panel was loaded
Writing Button	The writing panel is loaded when the button n is pressed	The panel has been loaded on the main window.	The panel was loaded
Home Automation Button	The automation panel is loaded when the button n is pressed	The panel has been loaded on the main window.	The panel was loaded
Game Button	The game panel is loaded when the button n is pressed	The panel has been loaded on the main window.	The panel was loaded
News Button	The news panel is loaded when the button n is pressed	The panel has been loaded on the main window.	The panel was loaded

Table 2 Test Cases - Loading the user panels

Comments

All test in this part ran as predicted, this was to be expected since we have used this function to see if other parts worked.

2.4. Writing Panel

2.4.1. Analysis and design

The typing functionality with the headset is essential a part of our application. The user should be able to write a text to the text box in the panel and control the keys with the help of the headset. As we described in the GUI section, the problem has occurred when we are using many buttons on the panel, then typing is very slow or delayed. On the other hand do we need to consider the relative position of the keys? Do we need to type all characters on the keyboards? Do we need to iterate all of the keys or only any section at a time? We asked ourselves those questions that helped us to better analysis and design the keyboard layout in order to increase typing speed.

Keyboard Layout

The main idea of the typing by headset is that the user will have certain time to invoke an click event on the button when said button is focused. For example when the user decided to type letter K, then he must wait while the button with the key K is focused and then he can invoke again the event click. So we are changing the focused on the next button by the timer and loop through all of the keys on the keyboard. There is also a button back to menu, so the user can exit from this the panel. This is the basic idea how we will control typing with the headset. We were testing an application and decided that the best length of time to leave button focused is 3s. It's not long and the user can easily work with that period. But after the first testing we figure out that typing is too slow. Because the typing speed also depends on how far apart the keys on a keyboard are. For example when we start loop through the keys and the user missed the letter K then we have to waiting till we have got focus again on the letter K. And this time is depending on number of keys (buttons) multiplied by how long a button is focused. Definitely we had to reduce this time.
First, we reduced the keys available on the keyboard. We have known that the user will not need to use all of the keys on the keyboard. Buttons as Shift, Ctrl, Alt, and Operators can be removed from the list (at least in this version, because later we can consider it again). Then we have sorted them to the sections. This means that we sorted keys according to the frequency of the letters of the in English text or the Danish text. This gives us the most frequently used letters and we split letters to the two groups. Less and more frequently used. The list of frequency letters in both languages can be seen in the Appendix E. According to *Concise Oxford Dictionary* the letters E, A, R are the most frequently used letters in English Vocabulary and in Danish language letters E, R,N. When we compared both of the list then we found out that both of the languages are using similar frequency of letters in text. Of course except extra keys such as Æ, Å, Ø. But we didn't have a problem to integrate them to the keyboard; we just moved the less used keys to the last row, See Figure 5.

The next thing, we arrange the numeric keys to the third section and functions will be in fourth section. We are not using too many functions, but we decided that we need the button clean text box, go to the menu, text to speech. We assumed that the functions will not be used such often, so it's good to keep them to another section. The result can be seen in Figure 4 on below.

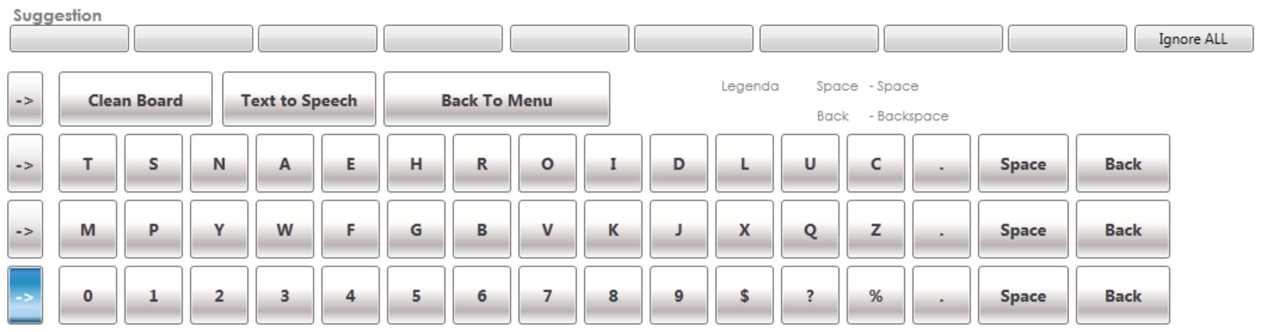

Suggestion
Ignore ALL
->
Clean Board
Text to Speech
Back To Menu
Legenda
Space - Space
Back - Backspace
T S N A E H R O I D L U C . Space Back
M P Y W F G B V K J X Q Z . Space Back
0 1 2 3 4 5 6 7 8 9 $? % . Space Back

Figure 4 Keyboard layout English

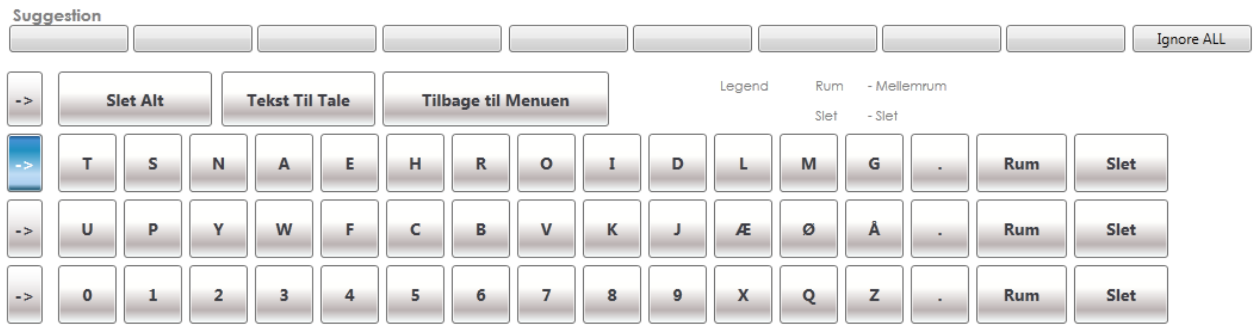

Suggestion
Ignore ALL
->
Slet Alt
Tekst Til Tale
Tilbage til Menuen
Legend
Rum - Mellemrum
Slet - Slet
T S N A E H R O I D L M G . Rum Slet
U P Y W F C B V K J Æ Ø Å . Rum Slet
0 1 2 3 4 5 6 7 8 9 X Q Z . Rum Slet

Figure 5 Keyboard layout Danish

The Text to Speech function is creating a spoken sound version of the text in the text box. So the computer transforms the text into speech in real time. We think that it can be very helpful for the user when he is typing text. You can also see that we put to every row the same keys such as dot, space and back. That is because we think that it will be easier for user reused these buttons in every row than still go back to the different section. We have though that these three keys can also be frequently used and specially Back (backspace). We assume that the user will decide, what a row he wants to use for the typing. Since the keys in row 2 are the most frequently used, then maybe just time-to-time he will go to the row 3. So now, we have 13 keys in a section instead 38 keys. We have increased typing speed but still is not enough, therefore we continued to improvement typing on a keyboard.

Now, the question is how the user will be selecting rows on a keyboard?

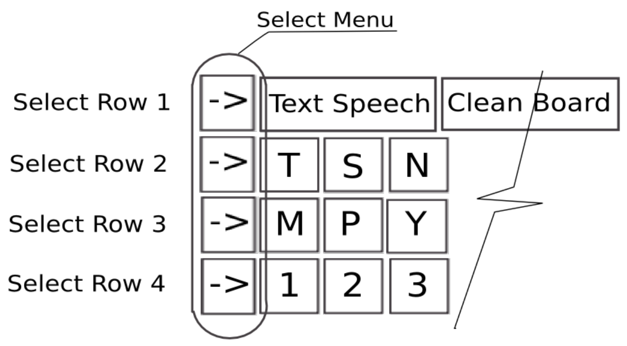
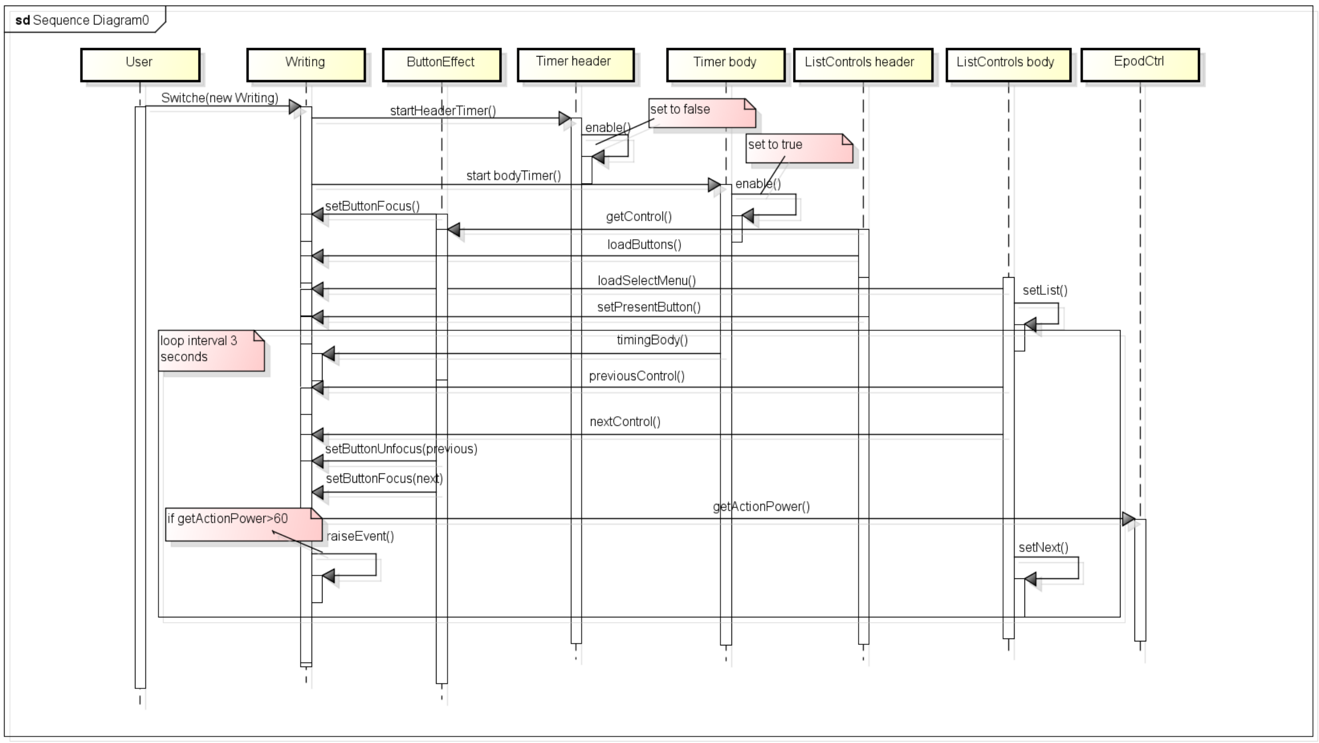


Figure 6 Detail of the keyboard

We implemented the select menu that is placed at the beginning of each row. Detail of the keyboard buttons can be seen in Figure 6. This select menu contains four buttons, each button one row. Basically we are iterating through buttons as we described at the beginning of this section and when the user will invoke event click on the button the application will start iteration in the selected



row. And again when the user wants to go the next row then he must wait till the select menu button is

Figure 7 Sequence diagram, iterate through the buttons

focused again and invoke the click event. This way, he will return to the select menu and application start loop through the select menu button again. An analysis is shown in the sequence.

We starting with the user when he invoked the click event in the header menu, and we overload the Switche static method that create the instance of the writing panel. When we are loading a new panel, first we need to initialize two timers, one timer for the header control and one timer for the body. The header timer is set as enable to false, because we don't want to go looping through the header buttons. Writing panel initialize the list of the controls. We are calling the two functions loadbuttons() and loadSelectMenu().

These methods are mapping all buttons control in the grid of the panel and adding to the list. The first method is mapping the header buttons and the second is loading selectMenu buttons and initializing to the list setList(). The ButtonEffect object is handling the focus on the button. When the button is selected, then we call a method setButtonfocus(). This method just highlight the focused button.

Because the timer of the body is enable true and the interval is set 3 second, then a timer delegate a method timingbody() every 3 seconds. This method is responsibled for the focusing buttons and checking a handset if the user changed cognitive state. So we are calling nextControl() and previousControl(). These methods return the buttons from the list of control than should be focused. Then we set as next button highlight (focused) and previous button return to the default state. Those functions are just making visual effect and the user can better see what button is focus right now. But during interval the 3 second interval method is checking the status of the headset if the value of the cognitive state was changed, if yes then a method eventRaising() is invoked that will trigger click event of the focused button. And at the end this timigBody method is called function setNext(),that will increase a index of the next button. Basically when the timer will delegate this function in the next iteration, then the current button became as previous and the next button will become current.

Same technique we are using for the all rows on a keyboard, except when the user selects the row then we load the list of the buttons from the specific row.

Spelling Checking

To increase the typing speed on a keyboard, we reuse the spelling-checking dictionary. When user starts typing a word, then automatically spelling checking is enabling to the true and then we just control spelling errors. We can see suggestion buttons in Figure 4. Spelling error generates the suggestions from the dictionary and we reuse these suggestions for the faster typing. We are just adding these suggestions to the buttons and a user can type all words instead one letter. Of course there is also option ignore all that will ignore the current spelling checking. When we compare speed of typing from the first version of the keyboard layout until the last one, we can see that we improved the typing speed very well. There are still possibilities how it could be improved.

Table 3 Typing speed for the various versions layout

Version	Number of keys	Time
1.	38 keys	38 * 3 = 114s
2.	10 keys (include select key)	10 * 3 = 30 s
3.	10keys (include suggestions)	30< s

Table is showing approximate time in worst case scenario, For example when the user invokes the key K on a keyboard then he must wait for the key K again this time.

2.4.2. Implementation

Adding letters to Text

We have two types of buttons. First group of the buttons are keyboard keys that are adding a letter to the text box and second group of the buttons are functions such as clean text box, go back to the menu.

Therefore we need to implement only two custom event methods that will be reused for every button. We called them OnClickKey and OnCLikOp.

In the xaml view we defined them as

For keys:

```
<Button  Click="OnClickKey" Grid.Column="1" Grid.Row="7" Margin="3 3 3 3" Name="buttonT"
Content="{DynamicResource bt_t}" Style="{DynamicResource FontStyle}" />
```

For function:

```
<Button Click="OnClickOp" Grid.Column="6" Grid.Row="6" Margin="0,6,30,3" Grid.ColumnSpan="4"
Name="btnBack"   Content="{DynamicResource back_menu}" Style="{DynamicResource
FontStyle}"></Button>
```

OnClickKey() Event is very simple, we need to just handle how we will add letters to the string. So we implement the handleLetter() method that will add letter to the text, as argument is passing letter. Textdisplay is the name of the text box. When we add letter to the text then we keep focus on the text box and displaying cursor at the end of text. It has just nice effect that we are writing directly to the text box.See methods on bellow.

```
private void HandleLetter(string l)
        {
            string str = textdisplay.Text;
            string newValue = str + l;
            textdisplay.Text = newValue;
            textdisplay.Focus();
            textdisplay.SelectionStart = textdisplay.Text.Length;
        }

protected void OnClickKey(object sender, RoutedEventArgs e)
        {
            Button bt = sender as Button;
            HandleLetter(bt.Content.ToString());

        }
```

Another function is a little bit complicate. First we need to distingue what button was pressed. Very button has tag property that can be use in our case. So we build an enum class Operator where we define all function on the panel.

```
public enum Operator
    {
        None,
        Clean,
        Delete,
        Back,
        Space,
        Read
    }
```

When the page is loading, we assign to the button property tag .

```
private void page_loaded(object sender, RoutedEventArgs e)
        {

            btnClean.Tag = Operator.Clean;
            btnBack.Tag = Operator.Back;
            btnread.Tag = Operator.Read;
            btnBackspace3.Tag = Operator.Delete;
            btnSpace3.Tag = Operator.Space;

        }
```

So we need one more function that we execute function and use it in the custom vent function OnClickOp()

```
private void OnClickOp(object sender, RoutedEventArgs e)
        {
            Button op = sender as Button;
            if (op.Tag != null)
            {
                Operator o = (Operator)Enum.Parse(typeof(Operator), op.Tag.ToString());
                ExecuteOperator(o);
                textdisplay.Focus();
                textdisplay.SelectionStart = textdisplay.Text.Length;
            }

        }

private void ExecuteOperator(Operator newOp)
        {
            string currentValue = textdisplay.Text;
            string newValue = currentValue;
            switch (newOp)
            {
                case Operator.Clean:
                    newValue = "";
                    break;
                case Operator.Space:
                    newValue = newValue + " ";
                    break;
                case Operator.Delete:
                    newValue = manipulateString(newValue);
                    break;
                case Operator.Read:
                    reader.Dispose();
                    if (textdisplay.Text != "")
                    {
                        reader = new SpeechSynthesizer();
                        reader.SpeakAsync(textdisplay.Text);

                    }
                    break;
                case Operator.None:
                    //
                    break;

            }

            op = newOp;
            textdisplay.Text = newValue;
        }
```

This function is mapping buttons in the grid

```
private void loadButtons()
        {
            if (header.countList() == 0)
            {
                foreach (Control ctrl in head.Children)
                {
                    if (ctrl is Button)
                    {
                        header.addControl(ctrl);
                    }
                }
            }

            body.SetList(loadSelectPanel());
        }
```

When we are loading pages then we need to initialize Timers for the control buttons. We will only show the code for the body timer and time Interval is variable equal to 3s.

```
private void startBodyTimer()
        {
            timerBody = new Timer(interval);
            timerBody.Elapsed += new ElapsedEventHandler(timingBody);
            timerBody.Enabled = true;
        }
```

Timer delegate method timingBody when interval elapsed.

```
private void timingBody(object sender, ElapsedEventArgs e)
        {


            this.Dispatcher.Invoke((System.Action)(() =>
            {
                Control ct = (Control)body.nextControl();
                Control previous = (Control)body.previousControl();
                // when the timer starting, then previous and actual control cannot be same

                if (body.nextControl() != body.previousControl())
                {
                    previous = body.previousControl();
                    effect.SetButtonUnFocus(previous);
                }


                ct = body.nextControl();
                effect.SetButtonFocus(ct);


                if (EpocCtrl.getInst().getAction.Power >= 60)
                {
                    int count = 0;
                    while (EpocCtrl.getInst().getAction.Power >= 60)
                    {
                        count++;
                        if (count >= 1000)
                        {
                            ct.RaiseEvent(new RoutedEventArgs(Button.ClickEvent, ct));
                            count = 0;
```

```
                                break;
                            }
                        }
                    }

                    body.setNext();
                }));
            }
```

From the code we can see that dispatcher maintains a prioritized queue of the work items for the timers. Then we call buttons from the listContol. NextButton() is the current button that is focused and use can see the button highlighted. Then we are checking from the Emotiv headset if the user wants to invoke event or not, When exceed 60 then we invoke the event on the focused button otherwise we are ignore it. As the last we need the set next iteration to the next button from the list Controls, so when the timer elapsed next event, then we will focused on the next button from the list controls.

2.4.3. Test

We wanted to do some unit testing for this part but because the output from the headset is unpredictable it wouldn't make sense to do this. Instead we ran some simple interface test just to see if the interface was responding to our commands, and to see if the methods we had implemented actually worked.

Description

This test case is going to make sure that all the keys and functional keys on the keyboard work properly.

Scenario

- Start the program
- Select the writing panel
- Navigating through the keys on the keyboard

Preconditions

The main window is loaded and running without problems.

Element	Description	Expected Result	Actual Result
ALL keys on the keyboard	The press keys on the keyboard	The letter will be typed on the text box.	passed
Clean button	The press keys on the keyboard	The text box will be cleaned.	passed
Text to Speech button	The press keys on the keyboard	The text will be speech from the text box.	passed
Backspace button	The press keys on the keyboard	The last letter from the text will be deleted.	passed
Space button	The press keys on the keyboard	The space will be added to the text.	passed
Suggestion buttons	The press keys on the keyboard	The suggestion text is added to the text box.	passed

Table 4 Test Cases – keyboard functionality

Comments

All test in this part ran as predicted, this was expected.

2.5. Engine

We did two different versions of the engine. The first one we did because we got stuck and had to move forward if we wanted to finish the application in time. For this reason we made a very basic version first and then got back to finishing the complete version.

2.5.1. Version 1

2.5.2. Analysis

Since we were spending a lot of time being stuck, and going nowhere with implementing the Training possibility inside our own application, because for the Training to occur first we need a Profile and a User. Unfortunately we couldn't get either one of them because the userID supplied on the emotive_onUserAddedEvent(this event was triggered as soon as the wireless dongle would be connected, therefore adding the default user template) couldn't be loaded to the Profile. We were guessing that this might happen because we only had access to the Developers Edition of the SDK. While going thru the material provided by with SDK and more specifically the .netEmotivSDKtest. We discovered one of the methods that we're essential initialize the buffer for receiving the data from the headset (EE_DataCreate). Later on after doing some searches in the community forum "why isn't It working, yet?" we found out, that like its mention a bit before, that the method was only in the Researcher Edition of the SDK.

2.5.3. Design

The design we used on this part was to get an instance of the EmoState from the Emotiv control panel and use it as part of our engine, so that we could use all the settings provided this way. We then used the events that where triggered to update a two global variables.

2.5.4. Implementation

We needed to define the events we want to listen for. In our case the only event we were interested in was the engine_CognitiveStateUpdate. Inside the event arguments we have the EmoState belonging to the Emotiv Control Panel so just a simple call for the methods that return the current action and power. Which then are stored in a global variable that would be used later. This a practical fix which helped us finally start moving somewhere with the project. We didn't give up on the training but just postponed it until we have something running and, we see if we have time.

As seen on the class diagram we only used the bare minimum to maintain the connection. After creating the instance of the emoState we connect to the headset. And then we start the thread to maintain the events that are raised.

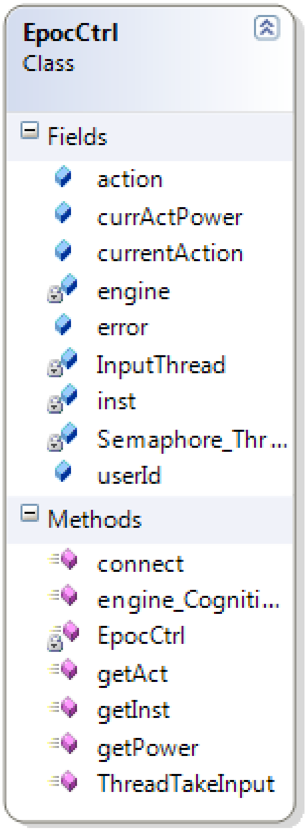


Figure 8 EpocCtrl Class Diagram

```
while (true)      {

                    engine.ProcessEvents();
                    System.Threading.Thread.Sleep(250);
                }
```
As you can see the event processing is very simple, we use the API to call the method for processing the event. It can be use in two ways to do its job, one way is to process all events in the buffer or if you are afraid that it will take too long you can set a timer that in millisecond, this function is just an overloaded method so it's very easy to implement.

As mentioned before the only event we used was the CognitivEmoStateUpdated, this event we used to get the cognitive action and power from the headset and set them to two global variables like so

```
            public void engine_CognitivEmoStateUpdated(Object sender,
EmoStateUpdatedEventArgs e)
        {
            EmoState es = e.emoState;
            currentAction = es.CognitivGetCurrentAction();
            currActPower = es.CognitivGetCurrentActionPower();
        }
```
We used the EDKdll to get these data and then it was just a matter of writing two getters for these variables.

2.5.5. Version 2

Analysis

As we enter version 2 of the engine we wanted some key features that we didn't have in the first one, these can all be tied into one entity called a control panel, the point of having our own control panel is that we want our program to be able to work alone without any help from the Emotiv software that comes with the headset.

This part of the program really was a big challenge since the documentation was very hard to understand since its made for C++ and no one bothered to update the documentation when they created the C# wrapper for the API.

Design

The engine consists of several different parts that all are needed to maintain the control panel. It based on a singleton pattern so we are sure that we always get the right instance of the engine. If we didn't go with this strategy we would get a new engine every time we made a call and all the information that we saved to the buffer would be unreachable.

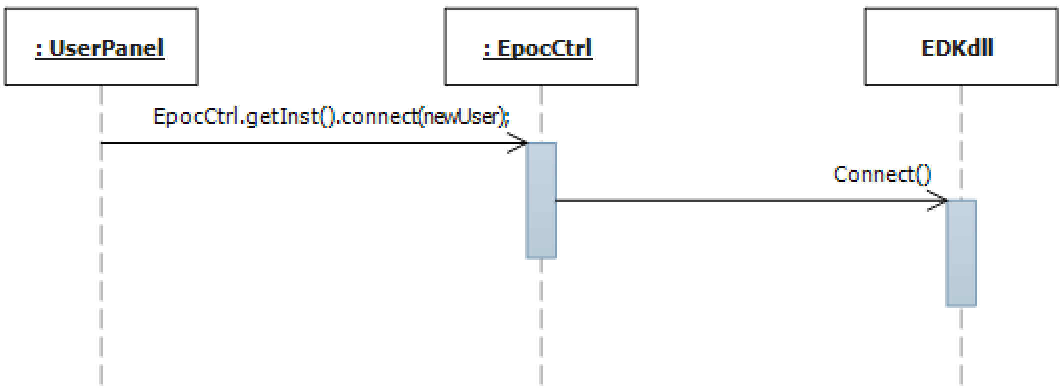


Figure 9 Connection Sequence diagram

Implementation

When implementing this part of the application we started to run some minor tests on how it would be possible to connect to the headset. We still had some projects that we did in the beginning and we used these to get started on the first draft. We found from the beginning that things was much easier now because we had a lot of experience with the API now and most importantly we had the understanding of how it works.

First thing was to make the actual connection, the problem here was that you only connect to the USB dongle and not the headset so you can actually get a connection without being able to get any data. To make the connection we needed to get the engine from the API and add the events that we wanted to handle so the instance can be created.

The next step was to make the function to connect to the headset, it's a simple method that connects and then starts the thread that is set to process the events. The reason why we start this thread here is that as soon as the connection is made we start getting events and this will tell us if something went wrong or if there are some problems with the headset.

When the above things are taken care of we needed to tell the system what data we wanted from the headset. This is done with the profile that has to be loaded into the engine. Here we can define what functions we want get from the headset and if it should be affective, expressive or cognitive. We decided from the beginning that we wanted to use the cognitive functions because the system is meant for people that can't use their body. So we implemented the loading of user profiles which is explained in further detail later in the report.

After going through these three steps we started to get some results from the headset so we decided to use another thread with a sleep timer to take care of the polling from the headset. The reason for using a thread and not a timer for this part is that the timer is tied into the STAthread and by experience it can make the GUI seem slow when we have to much running on the same thread, so because of performance issues we decided to do it this way.

When all was set with the communication we started on implementing our own training, this was very important to us because as soon as this was done we could leave the Emotiv control panel behind and not look back. Since we had a profile loaded into the system it was easy to see if the training we did was successful because when we loaded a base profile we didn't get any data out until we after the training and we could see that the debug information started to update.

Test

Since unit testing wasn't possible on this part of the program we decided to run black box tests instead. The problem is that all return values are unpredictable since it constantly changes and we would never be able to get the same result twice.

We wanted to test all functionalities, and for that reason we made a test plan so that we could keep track of what we expected to happen and what actually happened. It gave us a huge advantage that we did it this way. As you will see later on, it gave us something concrete to use during the testing period

2.5.6. The modules

The new engine isn't really a version 2 since its build on top of the old one but instead of connecting to an instance of the Emotiv software engine we create an object of our own engine and connect with the wireless dongle for the headset. The new engine can be divided into five different categories which is Event processing, data polling, profile management, training and state. To give a deeper understanding of these different parts we divided them and explained in the following section.

2.5.7. Event processing

Analysis

The event processing is the core of the engine. This is the part that makes us able to check what is coming next, and to do something accordantly. An event can be something like userAdded, which is triggered when the engine detects the dongle in the USB port. The way to use these events is to load them to the engine and then wait for the engine to trigger them.

Design

The way that the SDK for the Emotiv headset works is that everything is handled by events that gets fired when something new happen, since that is the way it's done we had to go with that architecture. The connection to the headset is maintained by the engine where we load the methods that should be triggered when an event is raised. Then the headset will tell the engine what the next events are and as they get triggered the application will move forward.

Implementation

The events we used can be seen in the code snippet beneath.

```
            this.engine.EmoEngineConnected += new 
EmoEngine.EmoEngineConnectedEventHandler(engine_EmoEngineConnected);
            this.engine.InternalStateChanged += new 
EmoEngine.InternalStateChangedEventHandler(engine_InternalStateChanged);
            this.engine.EmoStateUpdated += new 
EmoEngine.EmoStateUpdatedEventHandler(engine_EmoStateUpdated);
            this.engine.UserAdded += new 
EmoEngine.UserAddedEventHandler(emotivEngine_UserAdded);
            this.engine.CognitivTrainingStarted += new 
EmoEngine.CognitivTrainingStartedEventEventHandler(engine_CognitivTrainingStarted);
            this.engine.CognitivTrainingSucceeded += new 
EmoEngine.CognitivTrainingSucceededEventHandler(engine_CognitivTrainingSucceeded);
            this.engine.CognitivTrainingCompleted += new 
EmoEngine.CognitivTrainingCompletedEventHandler(engine_CognitivTrainingCompleted);
            this.engine.CognitivTrainingFailed += new 
EmoEngine.CognitivTrainingFailedEventHandler(engine_CognitivTrainingFailed);
```

These events seem fairly easy to do but has given us a lot of problems to get running because there really wasn't much help to get when it comes to creating a new control panel. We used a lot of time on it but we think that it is time well spend since it gave us a very good insight into how the headset works. After the events where set up we had to implement the methods for them. These methods range from setting a simple flag or setting the state to starting entire processes.

When all this is set up we are ready to start our thread that makes sure these events get fired. The way it works is that the engine holds a buffer where the different events get stored until the processEvent() method gets called, this method we call from a backgroundworker, which like the name suggests is a thread that runs asynchronic in the background. This is a good choice since we don't need it to be in sync with anything. The backgroundworker only has two jobs. The first job is to process the events to make sure that we have the most recent events fired. The second thing is to wait and give time for the polling to happen, this is very important because if this isn't done we would never get any output when we try to poll from the headset.

Test

Description

This test is going to make sure that all the events get triggered and they are called in the right sequence.

Scenario

- Start the program
- Connect the headset
- Read the returned results

Preconditions

All events should have a message box with debug information

Element	Description	Expected Result	Actual Result
EmoEngineConnected	The event that occurs when the engine is connected	Triggers when engine is connected	Triggered as it should
InternalStateChanged	The event that occurs when an error happens	Should output a stack trace	Could not create conditions to provoke error
EmoStateUpdated	The event that occurs when the EmoState changes	Should update the emotivState	Emostate got updated
UserAdded	The event that occurs when the dongle is detected	Should load a user	The user loaded as it should
CognitivTrainingStarted	The event that occurs when the Cog_Start action has been send	Should show a message	Got training start message
CognitivTrainingCompleted	The event that occurs when the training has completed	Should show a message	Got training completed message

CognitivTrainingSucceeded	The event that occurs when the training has been accepted	Should show a message	Got training completed message

Table 5 Event tests

Comments

All test in this part ran as predicted, this was to be expected since we have used this function to see if other parts worked.

2.5.8. Data polling

Analysis

In the first version of the engine we used an event to get the data from the headset, this was very unreliable because it took a lot of recourses to constantly take care of all the events that happened and wait for the special event that tells us that there is new output from the headset. Instead we decided that we would have a separate thread to get the information from the headset.

Design

The design of this part, is a simple thread that has a sleep timer, and then updates an object that holds the information from the headset.

Implementation

The data polling gave us a bit of problems the first time we tried to implement it because we wanted to handle it through the cognitive event that is given in the API. This we found to be very troublesome and in no way functional for use in the application. Instead we have started a thread in form of a backgroundworker. The way this thread works is that it first fetches an instance of itself so it can check if it has been canceled, then it goes to sleep for a fixed number of milliseconds, this is done so that the thread won't keep the entire system busy with this one task and lets other parts of the system work as well. After this wait the next step is to actually check if there is anything to get from the headset, this is done by a simple call to the headset and see if something is returned. After we have done these parts it's time to do the actual polling of data, this happens 4 times pr. second and updates into an object called Action, that can be read from any place in the program. In future versions we will implement bindings with the controls in the GUI part so that everything will happen much more smooth. With this new way to get the data from the headset we really made a great leap forward in the project because it's such an essential part of the control panel.

Test

Description

This test should check if there is anything returned from the method.

Scenario

- Start the program
- Load a profile
- Read the returned results

Preconditions

Headset should be connected and engine should be running.

Element	Description	Expected Result	Actual Result
Data polling	Check if data is polled	Activity bar should update	Activity bar updated as it should

Table 6 Data polling test case

Comments

2.5.9. Profile management

Analysis

Profile management is a tricky size to work on, especially because you have no idea what you are working with. Even though you know what is suppose to happen you have no idea if what you want actually happens. The reason for this uncertainty is that the profiles are written in binary files so you can’t just open one and check what’s in it. The only thing you can check is to see if the *.emu file is actually increasing in size.

One of the big problems that we found with the profile management is that during development it’s a lot easier to run with the composer instead of the headset, and even though the composer simulates the headset pretty well it still does not do what its suppose to do, so the files that you save is always just 1 kb. In comparison a brand new base profile with no training data at all is 17 kb and a profile with very basic training is around 170 kb.

Design

The way we decided to design the profile management is to use the build in events from the engine, so when you connect the engine to the headset an event is raised if the dongle for the headset is connected. This event then handles the loading or creation of the profile with a filename that is provided by the user.

In the diagram below you can see how the application communicates with the API.

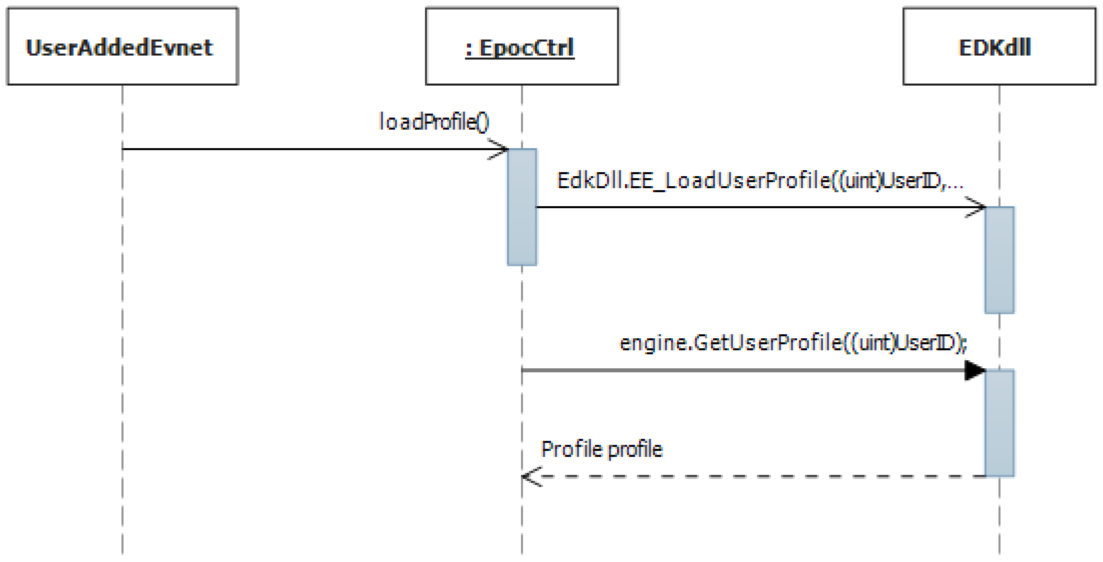

UserAddedEvnet
: EpocCtrl
EDKdll
loadProfile()
EdkDll.EE_LoadUserProfile((uint)UserID,...
engine.GetUserProfile((uint)UserID);
Profile profile

Figure 10 Sequence diagram for load user

When we load the profile we send a command to the dll file telling it what profile to load into memory, when this is done we send another message to get the profile bytes, which then returns us the profile. The last step is then to set the user profile as active.

Implementation

When the headset is connected the userAdded event will be raised and the method assigned to it will be called. This method will then call the correct method for either create or load profile depending on a variable that is either true or false. The reason for this way to implement the code is that because we use the event to trigger the action we can be 100% sure that everything happens in the correct order.

To get the profile you need three commands as shown beneath

```
EdkDll.EE_LoadUserProfile((uint)UserID, profilePath);
profile = engine.GetUserProfile((uint)UserID);
engine.SetUserProfile((uint)UserID, profile);
```

The first line loads the profile into memory and assigns it an id number that can be used later for manipulation. The second line gets the profile bytes and loads them into an object of type profile which is just an intPtr with some extra methods. The last method call sets the profile that we just loaded and sets it as active in memory. This is the way to go if you want to have a fully working profile on your system, but that's not always the case. If the profile you are loading is just a base profile you also need to tell it what you are interested in using, you do this with the setActiveActions() method. This method is very tricky to figure you, again because you can't just open the profile and check what you are changing. These active actions are being calculated by inputting an integer value that is extracted by the number of placement in the action has in the enumerator. The engine can handle a max of 4 active actions at a time so you have to decide what actions you want to focus on.

When you want to create a user it's almost the same as loading a base profile because here we also want to set the active actions. The only difference is that instead of getting the profile bytes we create a new object that the API automatically set as a base profile. As you can see below the code look almost the same

```
profile = new Profile();
engine.SetUserProfile((uint)UserID, (Profile)profile);
engine.EE_SaveUserProfile((uint)UserID, profilePath);
```

Test

Description

This test is going to test the overall functionality of the engine.

Scenario

- Start the program
- Press the buttons
- Read the returned results

Preconditions

Dongle for the headset should be plugged in.

Element	Description	Expected Result	Actual Result
Load User Profile	Load a user profile	Activity bar should begin to update	User loaded and data was translated
Create User Profile	Create a new profile	File should be created	File got created and held data
Save User Profile	Save a user profile	Profile size should rise	Profile got saved to the file

Table 7 Manage user test cases

Comments

All test ran as expected

2.5.10. Profile Training

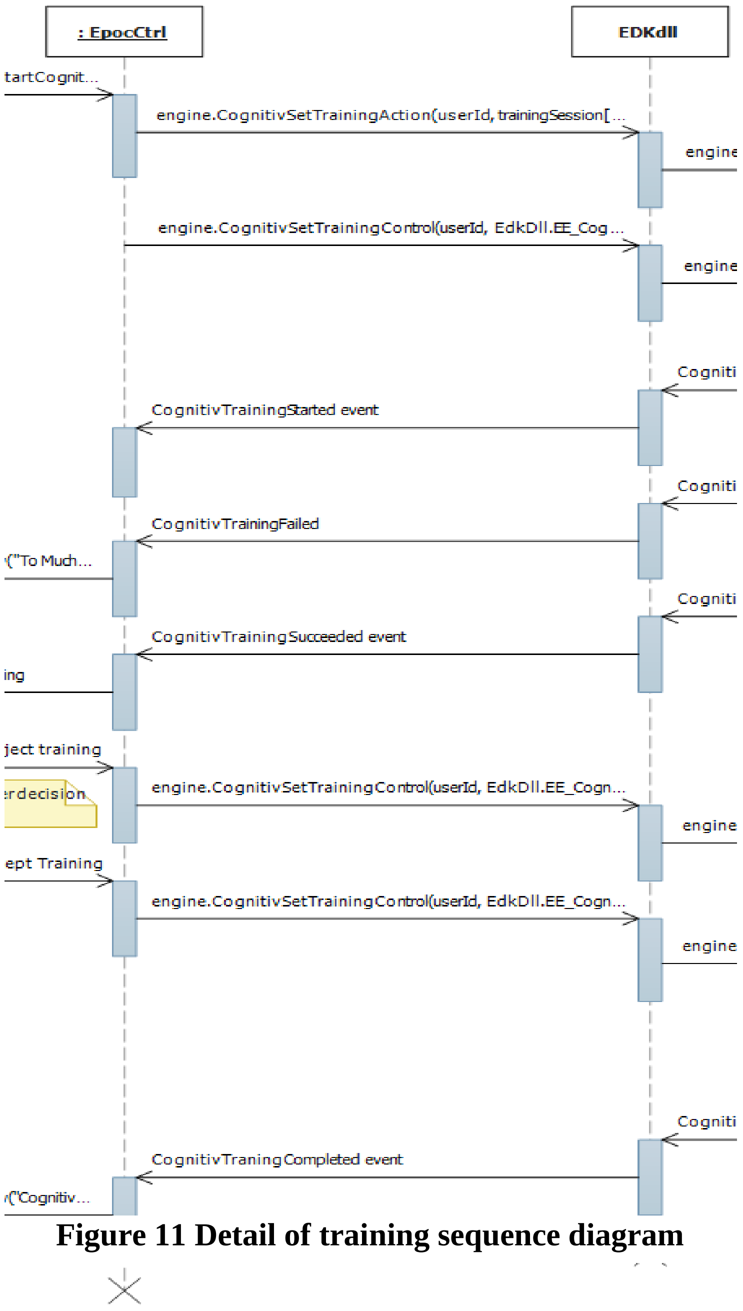
Analysis

To be able to use a profile it is necessary to train the actions you want to use, because a base profile doesn't contain any information on what the brain patterns look like and therefore can recognize them. The actions we want to work with are neutral which is the relaxed pattern and the push pattern. The pattern is different in everybody so the headset needs this data to be able to trigger when you want something to happen.

Design

The way we decided to implement the training is to call a method that sets the necessary parameters and then starts the training process that basically is three steps. The first is started, the second is completed and the last is succeeded.

The diagram shows a snippet of what is going on then the training is started. The entire diagram can be seen in the appendix G. This part of the diagram shows the



: EpocCtrl
EDKdll
tartCognit...
engine.CognitivSetTrainingAction(userId, trainingSession[...
engine
engine.CognitivSetTrainingControl(userId, EdkDll.EE_Cog...
engine
Cogniti
CognitivTrainingStarted event
Cogniti
CognitivTrainingFailed
("To Much...
Cogniti
CognitivTrainingSucceeded event
ing
ject training
rdecision
engine.CognitivSetTrainingControl(userId, EdkDll.EE_Cogn...
engine
ept Training
engine.CognitivSetTrainingControl(userId, EdkDll.EE_Cogn...
engine
Cogniti
CognitivTraningCompleted event
("Cognitiv...

Figure 11 Detail of training sequence diagram

interaction between our application and the dll file edkdll that holds the methods for communicating with the headset. It shows how the communication is and how it should be handled.

Implementation

After the profile has been loaded you can choose the option to train the different actions, this is necessary for the engine to better map the brainwaves that will appear when you do the different movements. When you train the actions it is important that you relax and let your brain do the work, this means no clenching your face or biting your teeth together [3]. The reason for this is that it will send muscle signals that will be recorded, and if you do the same for every signal the training will end up to much alike no matter what you do.

The way we chose to set it up is to have two basic functions which are neutral and push. The way to go about the training is to first make sure that the action is active, this is easiest done by just setting them again. After it's done you select what function you want to train and after that, you start the training session as follows.

```
engine.CognitivSetTrainingAction(userId, trainingSession[session]);
engine.CognitivSetTrainingControl(userId,
EdkDll.EE_CognitivTrainingControl_t.COG_START);
```

When the training has been started three events will be triggered, the first one is training started. As soon as that has been started there will be a eight second window where the recoding of the brainwaves will take place, after this timeframe the user will be asked if they managed to focus on the function that is in training, if yes it will be saved if no it will be discarded. If the training fails an event will be triggered and we had to make a decision on what to do with it. We decided that the best thing was to let the user restart the training instead of letting the application take care of it by itself. We made this decision because there might be some things that needs to be adjusted and then it would just be a waste of time to run the training again.

Test

Description

This test is going to check if the training will actually add some information to the profile.

Scenario

- Start the program
- Try the functions
- Check the file sizes

Preconditions

Headset should be on and probably connected to the engine

Element	Description	Expected Result	Actual Result
Train Neutral Action	Start and complete Neutral training	File size should rise	File size incremented
Train Push Action	Start and complete Push training	File size should rise	File size incremented

Table 8 Profile training

Comments

All tests ran as expected

2.5.11. EmotivState

Analysis

One of the key elements in the core of the program is the state. The state holds the important maintenance information about the system, information like how the connection to the headset is or the battery level. It's also the state that holds the electrode connection status or the actual information from the headset.

Design

Because the emostate is a pretty big class with a lot of methods we have decided to make our own that only holds the things that we need. The things that we implemented are the methods for calling the cognitive action and the cognitive action power from the headset, and the methods for getting the variables for the headset information. The class acts like a normal object and interact with the API directly. As the class diagram to the right shows the EmotivState class has been designed to tie all the headset communication methods together in one class.

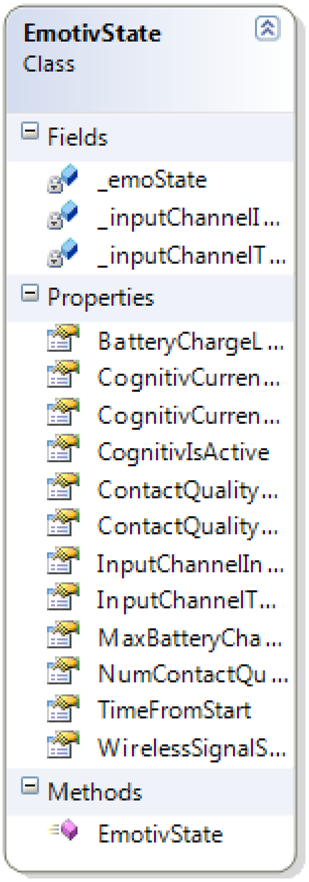


Figure 12 EmotivState class diagram

Implementation

The way that we implemented the emotivstate class, that we called our own version of the emostate is to take a clone of the actual instance of emostate. This way we are sure to get all the correct information transferred while we have the information in the format that we want. When the first event for the emostate updated is triggered we have all the information for creating our state and we do that as follows.

```
emoState = new EmoState(e.emoState);
emotivState = new EmotivState(emoState);
```

as you can see we first create the actual state and then clone that one into our state by taking it as a parameter, its then set to a global variable so it can be accessed from anywhere in the class. The good thing about this way to do it is that if there are any changes to the state we will right away get it updated.

The main reason for us to make our own state is to know what is actually going on inside it, the version in the API is written in unmanaged code that we can't use in our WPF so it's set in a c# wrapper and it's impossible to find heads or tails in it. Therefore we found that the optimal solution would be to do it this way instead of having to work with integer pointers and trying to get them wrapped in the right way.

Test

Description

This test is going to check if we get some input from back from the headset.

Scenario

- Start the program
- Read the returned results

Preconditions

Headset and engine should be on and connected.

Element	Description	Expected Result	Actual Result
Get Wireless signal strength	check if there is a signal from the headset	Full signal from the headset	Full battery status
Get battery level	Check the battery level	Get value indicating battery level	Got integer value 3 indicating battery level
Get Electrode Connection State	Check is all electrodes get input	Get list of values indicating electrode status	Got full list of electrode connection status
Get Cognitiv Action	Check output from headset	Get cognitive action enum	Action updating as it should
Get Conitiv Power	Check output from headset	Get cognitive power value	Power updating as it should

Table 9 EmotivState tests

Comments

Everything passed as expected

2.5.12. Updates

For future updates we want to implement the INotifyPropertyChange interface because that will give us the possibility to bind the WPF controls directly to the headset, but we hit a problem since we are using a singleton for the engine and that can’t be loaded in a proper way.

We also want to improve the training in future versions of the application, it works fine now but we want give the user an opportunity to use more different functions. The problem at this point is that it’s hard to handle more than one function and until we cover that part we don’t want to add to many cognitive functions. We also want to add some animation to help training but since none of us are strong in animation we have decided to leave it for now.

2.6. User Panel

Analysis

Now that we had our own Engine and we had to build an interface similar to the Emotiv control panel. The elements we considered key were: the ability to create and load a profile on which you could then train. Also the training, the connectivity of the pins, so you can see when you positioned the headset correctly, the wireless and battery signals and last but not least the current action and level registered by the engine.

We decided to make the following classes for an easier control of the system. The Pin class represents a Headset Node. The check value holds the type of signal in string(“No Signal”, “Good Signal”, “Fair Signal”, “Poor Signal”, “Very Bad”), the status value holds the color corresponding to the signal (White, Green, Yellow, Orange, Red). When this class is instantiated with EE_SignalQuality_t it will automatically set the check and status to the appropriate values. In the battery the status again represents the color depending on the level.

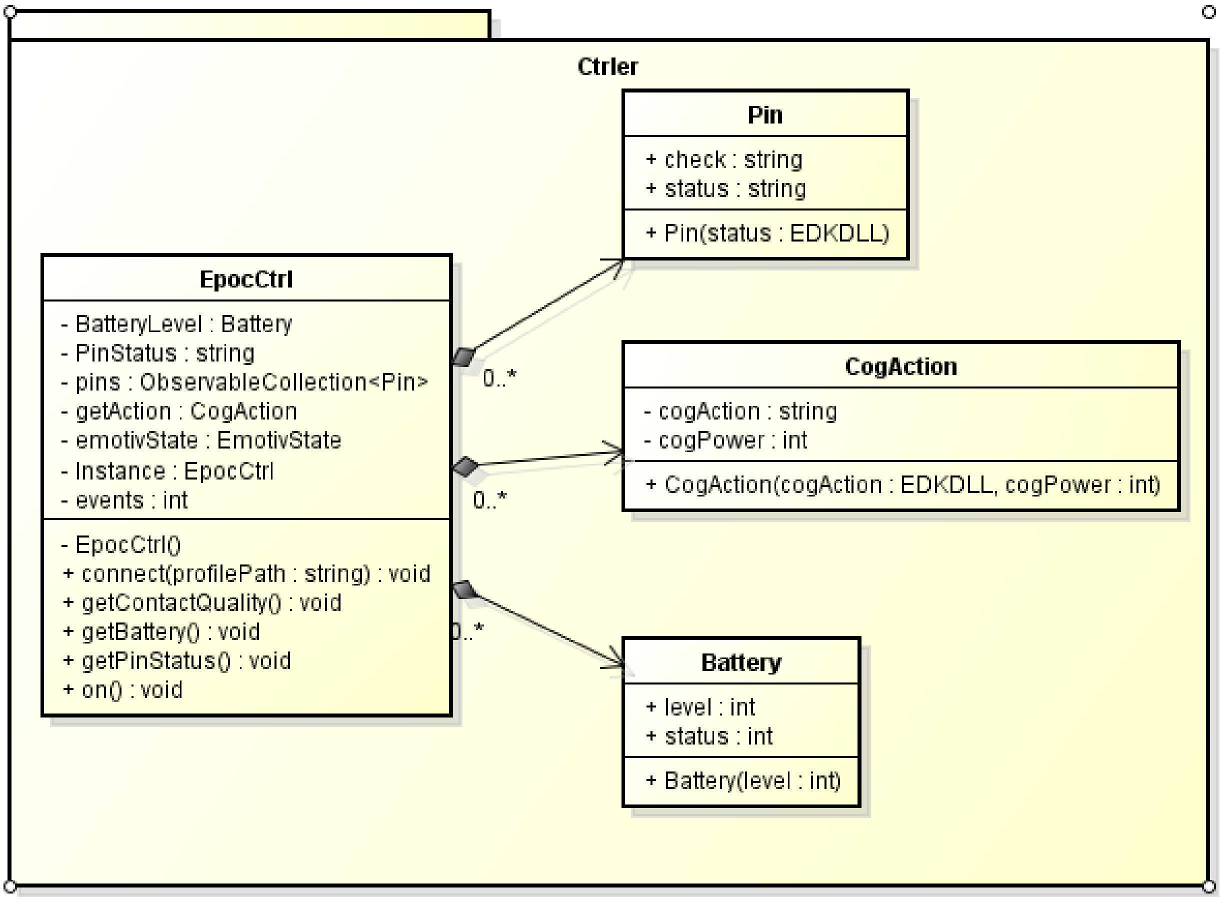


Figure 13 Class diagram of first controller

Design

The profiles manipulation consists of 2 buttons (Create and Open) on either click of them it will start looking for the headset and establish a connection and handle the profiles. As soon as a connection is made the UI comes to life.

In the training department we have one button which will notify our controller (EpocCtrl) to start the training with the action selected in the comboBox. In the same time it will set the visibility of the timerBar to Visible and start a timer on the controller which will be active for 8 seconds with an interval of 1 second and inside the the tick event it will count++ a value which is bound to the timerBar.Value. This way the user can keep track of how much time is left of the current training session.

Available on all pages are the 2 progress bars indicating the current action power and the battery life plus the 2 rectangular allGood and WirelessSignal, because of their obvious importance. The purpose of the allGood rectangular is to keep you notified with the connectivity of the Pins when you navigate away from the UserPanel.

Implementation

The initial idea for showing the connectivity of the pins was to have a list with the pin number and a rectangle next to it with the respective color. But we decided to take it overboard and we drew over an image of a head, 16 ellipses (circles) where each node should be. The idea was good but each ellipse was a field (therefore we had 16 fields and on each specifying the pin position inside the canvas) and for each field we had to make a property to bind to (resulting in 16 properties). This is just bad programming so we researched it a bit more and came up with a solution. Inside WPF the ListBox acts like a generic List. The way we use it is that we make a List of Ellipses and then we can easily bind it to our ObservableCollection.

```
<ListBox Style="{DynamicResource objectList}" ItemsSource="{Binding pins}">
                <ListBox.Background>
                    <ImageBrush ImageSource="../img/emotivCMSDRL.png" />
                </ListBox.Background>
            </ListBox>
```

So since we are not defining each ellipse for each electrode individually anymore, and therefore not setting the position for each electrode, we had to make a “template”. So we made the Pin class work for us by adding 2 fields top and left (representing the distance from the canvasTop and Left) so we can bind them to our Ellipse and in the constructor we also added a parameter as an integer (index). Since we refresh all of the pins at once and not individually (the data from headset is of type EE_ContactQuality_t[] <-list) it’s not hard, for each time we instantiate the pin to also add the index. Next we have the PinCanvas class which is a static class in which we hardcoded the position values for each pin. There are 2 lists of type integer with the corresponding locations on the canvas for the pins and get methods in which you supply the index you need. Than on the initialization of each pin we query the PinCanvas with the index so that we easily can bind or Pin-s to our view.

```
<UserControl.Resources>
        <Style x:Key="objectList" TargetType="ListBox">
            <Setter Property="ItemTemplate">
                <Setter.Value>
                    <DataTemplate>
                        <Canvas>
```

```
                             <Ellipse Width="25" Height="25" Stroke="DarkGray"
StrokeThickness="2" Fill="{Binding check}" Canvas.Left="{Binding left}" Canvas.Top="{Binding
top}"/>
                         </Canvas>
                     </DataTemplate>
                 </Setter.Value>
             </Setter>
         </Style>
     </UserControl.Resources>
```

We have 5 types of Pin signals so to each pin we assign a value from 0 to 5, depending on how good the signal is. Then we add it together and next calculate the average. Depending on the result we set the color of allPinStatus to (3<=Red<24<=Orange<40<=Yellow<56<=Green<=64)

All of the above are bound to properties inside the EpocCtrl. Here is the reason why we chose to add the EpocCtrl instance as an application resource. This way when we add the "EpocCtr" in the MainWindow DataContext we will have the model available for all the Pages(User Controls/Views).

```
     public partial class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);
            Resources.Add("EpocCtrl", EpocCtrl.Instance);
            Resources.Add("NewsViewModel", NewsViewModel.Instance);
        }
    }
```

The EpocCtrl handles all the events from the Engine and notifies the UI accordingly. In the case of the Battery, WirelessSignal, allPinStatus, currAction and Pins everything happens pretty much the same. Each time there is an event we query the headset for all the data and update the properties inside the EpocCtrl to their new values. As soon as each property updates it notifies the UI to also update. For the Battery we made a counter and each time it can be divided by 20 it updates the battery (1 time after others were update 20 times). We do this because the battery won't change its status very often.

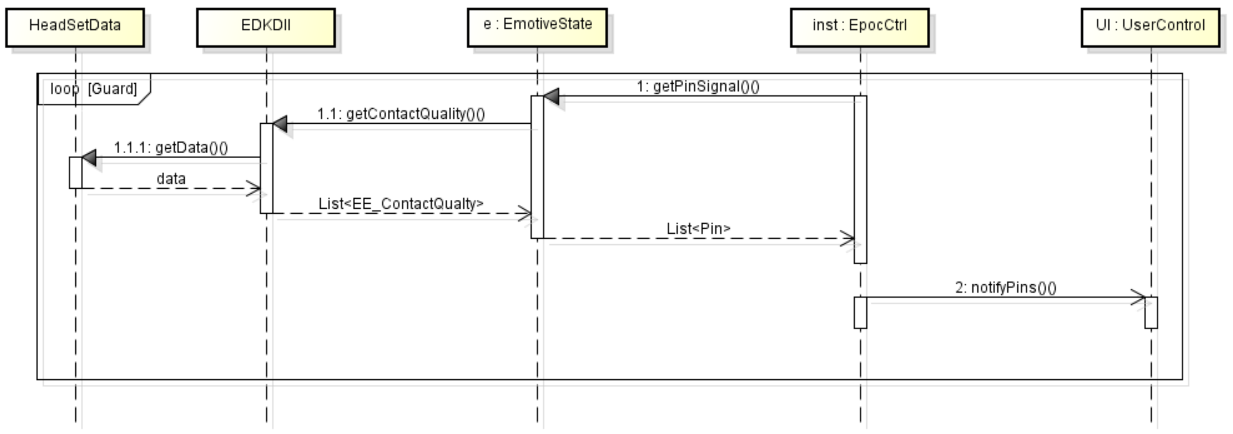


Figure 14 Sequence diagram headset data

Test

We didn’t run any tests in this part since this already has been tested in the engine.

2.7. Options

2.7.1. Analysis and design

When we started analyzing an application, we figured out that it would be better when we separate all settings to the option panel. First, it is more common for the user, second maybe the user will not need to change setting such often. Although we have only two settings language and style of the buttons now, we expecting that size of the parameters in the option panel will be increasing.

Language and Style

The Multilanguage application is becoming more competitive, more convenient for the end user and every organization. Design and development such an application doesn’t have to be difficult. WPF support many techniques that can be used for this challenge. Each control has a property named Content that can be dynamically bound from the resource file. Similar principle is with the style of the controls. If we want to set different a setting for the button, then we need to create a recourse file and define style that we want to use. The recourse file is a xaml file that will keep all information that are dynamically loading to the controls.

We designed three different styles that will be used for the buttons and every language will be having its own resource file. So when we want to change language, then we need just add resource file the ResourceDistionary and Language or style will be changed. Each new user that is created has default settings that are stored to the xml file when we close the application. When the user changes these settings, then the application will update them dynamically and save to the xml file.

Implementation

Examle of DK.xaml resource file, where we used the name of key back_menu, so a control with this key will be binding the text between brakets.

```
<system:String x:Key="back_menu">Tilbage til Menuen</system:String>
<system:String x:Key="lb_language">SPROG:</system:String>
```

Then in the view model we can call content dynamically from the resource file.

```
<Label Content="{DynamicResource lb_language}" Name="labelLangue" />
```

Resource file is added to the ResourceDictinary when we are loading the panel. We need just to call the language or style of the logged in person.

```
public void actualStyle()
        {
            ResourceDictionary dictionary = new ResourceDictionary();
            dictionary.Source = new Uri("pack://application:,,,/MBoard;Component/Style/" +
person[index].StyleNum);
            Application.Current.Resources.MergedDictionaries.Add(dictionary);

        }
```

Similar way we can add the style to the resource directory.

```
public void actualLanguage()
        {
            ResourceDictionary dictionary = new ResourceDictionary();
            dictionary.Source = new Uri("pack://application:,,,/MBoard;Component/Style/" +
person[index].Language);
            Application.Current.Resources.MergedDictionaries.Add(dictionary);

        }
```

Test

Description

These test cases are going to test if we changed language and style on all controls.

Scenario

- Start the program
- Go to the Option Panel
- Select language and select font size

Preconditions

The application is running and we have default settings on the panel.

Element	Description	Expected Result	Actual Result
Danish Language button	the button is pressed	all controls must change a language to update	Passed
Font Size 16	the button is pressed	all buttons change the font size to 16	Passed
Font size 18	the button is pressed	all buttons change the font size to 18	Passed

Table 10 Test Cases – Testing Options

Comments

All tests ran as we expected.

2.8. Store Data to the XML file

2.8.1. Analysis and design

Storing the data in an application can be done various ways. One of the techniques is storing the data to the XML file. XML (text-base markup language) is simple language that is human readable and machine-readable. It does easy identify the data that are stored in the file. We decided that XML file fit for our application. We have a list of the persons that we want to store with their settings and statistics. When we are loading or closing the application, the data should be read or write from the xml file.

2.8.2. Implementation

In the main window we call event Window_closing, where we writing all persons from the list to the xml file.

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
        {
            service.setDefaultPerson();
            xml.writeXMLPersons(service.getListPersons());

        }
```

The method is passing the list of the person as argument. First we are calling XML writer and initialize directory to the xml file. Then create start element and foreach loop start writing every person to the file.

```
public void writeXMLPersons(List<Person> list)
        {
            try
            {
                XmlWriter writer = XmlWriter.Create("../../App_Data/Person.xml");
                writer.WriteStartDocument();
                writer.WriteStartElement("Person");
                foreach (Person p in list)
                {
                    writer.WriteStartElement("Person");
                    writer.WriteElementString("Name", p.Name.ToString());
                    writer.WriteElementString("Language", p.Language.ToString());
                    writer.WriteElementString("Style", p.StyleNum.ToString());
                    writer.WriteStartElement("Statistics");
                    writer.WriteElementString("Wins", p.Statis.Wins.ToString());
                    writer.WriteElementString("Lost", p.Statis.Lost.ToString());
                    writer.WriteEndElement();
                    writer.WriteEndElement();
                }

                writer.WriteEndElement();
                writer.WriteEndDocument();
                writer.Close();
            }
            catch (XmlException e)
            {
                Console.WriteLine(e.Message + "  Writting XML");
            }
        }
```

When we want to read the person from the list, we need to do reverse way. First, we load XML document from the directory and then load inner text from every node to the Person object, each person is added to the list of the persons.

```
public List<Person> readXMLPersons()
        {
            List<Person> temp = new List<Person>();
            try
            {
                XmlDocument doc = new XmlDocument();
                 doc.Load("../../App_Data/Person.xml");

                XmlNodeList nameNodes = doc.GetElementsByTagName("Name");
                XmlNodeList langNodes = doc.GetElementsByTagName("Language");
                XmlNodeList styleNodes = doc.GetElementsByTagName("Style");
                XmlNodeList winsNodes = doc.GetElementsByTagName("Wins");
                XmlNodeList lostNodes = doc.GetElementsByTagName("Lost");

                for (int i = 0; i < nameNodes.Count; i++)
                {
                    temp.Add(new Person(nameNodes[i].InnerText,
                                         langNodes[i].InnerText,
                                         styleNodes[i].InnerText, new
Statistics(Convert.ToInt32(winsNodes[i].InnerText),
Convert.ToInt32(lostNodes[i].InnerText))));

                }
            }
            catch (XmlException e)
            {
                Console.WriteLine(e.Message + "  Reading XML");
            }

            return temp;
        }
```

2.9. Home Automation

2.9.1. The code

Analysis

The home automation that we wanted to implement was to give the user of the system the ability to control bed and lights in the room where the user is positioned. The function of this module has been made as simple as possible and involves two buttons that are picked automatically so that the user can press the button when it’s selected by the system.

Design

The home automation GUI consists of two buttons to emulate the conditions that we need, so we can either adjust a bed up and down or turn light on and off. Behind the scenes it’s a little more elegant. When the page is loaded a backgroundworker and 3 timers is started. The reason for using a background worker here is that it’s good for running asynchronous callback functions so that it won’t interfere with the STAthread and the GUI won’t get unresponsive.

The backgroundworker is used to do the actual clicking of the buttons. It does this by raising the event of the button being clicked and in that way bypasses the actual need to click a button. We then use three timers where two of them cycles through the buttons. One is for the top panel and one is for the buttons needed to navigate the controls for the automation, in this part we are only going to focus on the automation controls. The last timer is used to handle how long the output should be on when you press a button.

Implementation

The timers are the key to moving to the next button, and keep track of where the previous is. We run this in 3 second intervals because after testing we found that this interval gives good time for even beginners with the headset, to focus and get the button pressed. We tried to use threads for this part but we found that they very hard to control. If we ran it in the foreground we had problems that it interfered with the STAthread, and if we ran it in the background we couldn't really be sure when it ran. So after testing the first version we went with timers.

The backgroundworker runs the entire time the page is loaded, it runs with an interval of 500 ms and simply checks to see if the user has activated the headset to a value above the threshold we have decided to go with. The way we did this is to check the value for a fixed number of iterations to be sure that it's not just a spike in the system and then press the buttons. We tried in a previous version to do this in the timer that iterates the buttons but this failed in tests because it required the user to have the headset high when the button first got focused and that resulted in a lot of presses on the button after the intended.

After intensive testing with different combinations of threads and timers we came to the conclusion that if we wanted the most accurate system with the least delay we has to go with the backgroundworker and the three timers.

The last timer is only used to control the time that the RS232 port sends out a signal. The reason why this is necessary is that we are using the RTS and DTR signals as output and the length of the signal is very different so instead of using iterations to try and get them to be about the same length we have decided to implement a timer that runs for 3 seconds and the raises a flag that closes the port.

When we want to press a button we wait for the correct button to be highlighted and then imagine that we press it. If it is done correct the backgroundworker will trigger the event that then presses the button. In the following code snippet you can see how it's done.

```
private void pressButton(Button button)
        {
            if (button.Dispatcher.CheckAccess())
            {
                button.RaiseEvent(new RoutedEventArgs(Button.ClickEvent, button));
            }
            else
            {
                PressButtonCallback p = new PressButtonCallback(pressButton);
                button.Dispatcher.Invoke(p, new object[] { button });
            }
        }
```

As seen in the snippet we use the dispatcher to check the access to the control that we want to interact with and then raises the event accordantly with the dispatcher if necessary. The reason that we need to do this is that the controls are owned by the STAthread and that makes it impossible to interact with it in a normal fashion. This was why we wanted to use the timer for this function since its part of STAthread and therefore can access it directly.

In the activity diagram beneath we have a visualization of how the module works.

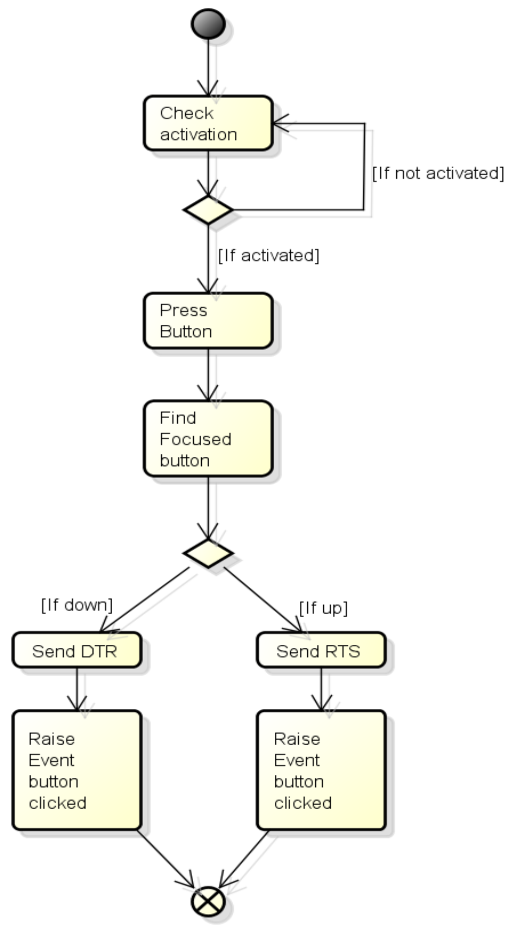


Figure 15 Activity diagram of home automation

2.9.2. The board

Analysis

The board is made as a proof of concept since the people from Invacare didn't live up to the agreement that we made with them and didn't send the control unit for a hospital bed. The circuit is build on the Max232 line driver

After some initial testing for what could work and what couldn't we turned our attention to how we could manipulate the signal, and after some time we decided to go with the Maxim 232 IC.

Maxim 232 IC is a line driver that can convert the signals that come from a RS232 serial port and turn them into TTL signals that can actually be used. The board takes input from the port and a 5 volt supply voltage to turn the minus voltage into positive voltage that can actually be used.

Design

The design of this circuit is very basic. It consists of the Max232 IC the four capacitors as described in the Maxim 232 user manual, two diodes to protect the rs232 port from feedback and tow simple gates created out of two resistors and one NPN transistor.

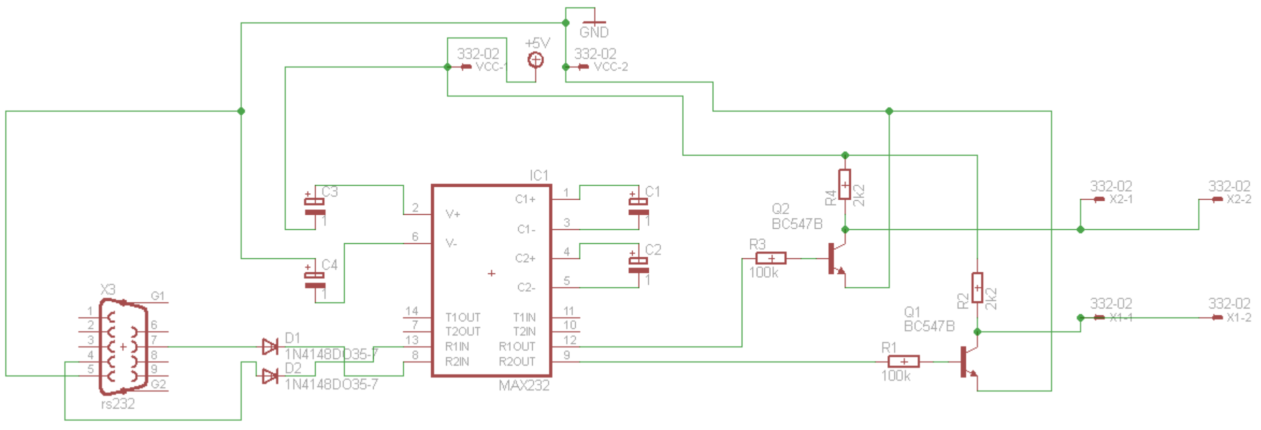


Figure 16 Diagram of the Mboard circuit

As seen in the diagram above it’s a simple circuit that uses the output from the RS232 port to control the two gates that will let the supply voltage through to the output pins. The reason for the pins to be in pairs is that it will be easy to attach them to an H-bridge if we want motor control.

The two diodes are placed in front of the Rts and Dtr pins on the rs232 port for the simple reason of protecting the port in the computer from feedback. The chance of feedback happening is really small if there at all we decided to have it anyway.

The finished design of the board can be seen beneath.

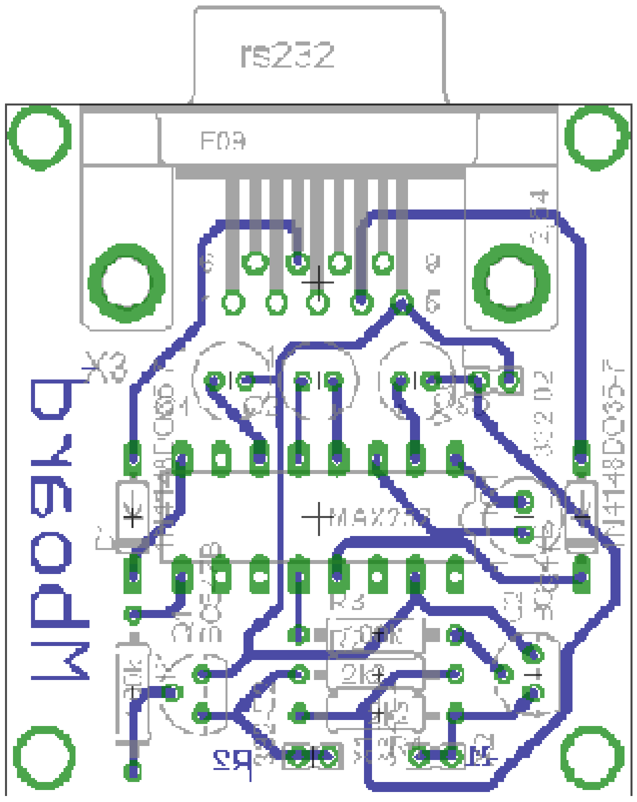


Figure 17 Mboard circuit layout

Implementation

When we started this part we hoped that it would possible to connect the port directly to a motor without too much work in electronics, but since the output of an rs232 port is -15V – +15V we had to add a line driver in between the port and the motor.

After the testing and prototyping was done it was time to actually make the board. For this task we used eagle since it's easy to go from drawing to board with this software. After the board was made and the soldering had been done it was time to test it, we were fairly confident that it would work since we made the board after the prototype but it didn't go that easy this time. Because eagle doesn't add vcc and gnd in the schematic we had to add it in the board file and of cause someone forgot to add vcc. This was fixed with a small connection from the voltage supply to vcc on the IC.

First way the code was implemented was to use the Trx pin on the rs232 pin running on one port on the max232 IC. This gave us 2,5V and 5V on the output pin, and then the plan was to spilt this to two different inputs and control a motor this way. After a couple of trial runs on the breadboard we decided that having two different outputs would be better and give us more control of how it can be used. The next problem was that we had to find two pins that we can use for this input. We decided to use the Rts and the Dtr pins because even though the time for them to run is different, they act in almost the same way, so it's just a matter of adding some timers with the same duration so they will seem to be acting the same way.

The way it works is that as soon as the IC is on and if either Rts or Dtr pin changes the current will rise and the gate will open. This will then result in the supply voltage to be let out and the output of the board will be 5 volts. We are sure that there are other and better ways to make a board that works as good as this one but since our project is about creating an application we don't want to spend too much time on this part. Since the board is meant as a part of a bigger system we didn't worry about supply voltage. This would probably come from a large battery, but since we only use this part as proof of concept, it seemed as a good solution to just use a 5 volt external power supply and not spend time on building something our selves.

Test

The testing started very early in this part of the project since we didn't have any experience in using a rs232 port in this way, we knew that we had to get some kind of signal out and a serial port seemed as good as any.

The first test was to check what we could actually get out from the port and by reading the specifications on the rs232 standard it was clear that we had to use Tx, Rts or Dtr and the test was very successful and we found out that we had to use either a line driver to get the correct result or connect it to a microprocessor. At this point in the project we decided that it would be a bad idea to start adding an embedded system to the project, since that would be a huge extra workload that we weren't sure we would have time for, and then the decision was easy. We went with the line driver.

After getting the Maxim 232 port we started to build a prototype on a breadboard, we started to test the output of the Tx port but the TTL signal we got was between 2,5 and 5 volts and it made it more complicated to use than it had to, so after discussing it in the group we decided that it would be better to use two signals so that we could have either 0 or 5 volts. This would be easier to control and easier to construct.

When it was all done it was time to do the black box testing. Since we don't get any return value on a print board we connected the output to a voltmeter and recorded the output.

Description

We test to see what the output from the circuit is.

Scenario

- Start the program
- Connect the board
- Read the output

Preconditions: A RS232 port should be available

Element	Description	Expected Result	Actual Result
Press up button	Get output on the output terminal up	5v output	5v output
Press down button	Get output on the output terminal down	5v output	5v output

Table 11 Press button test cases

Comments

Everything went as expected

2.10. News

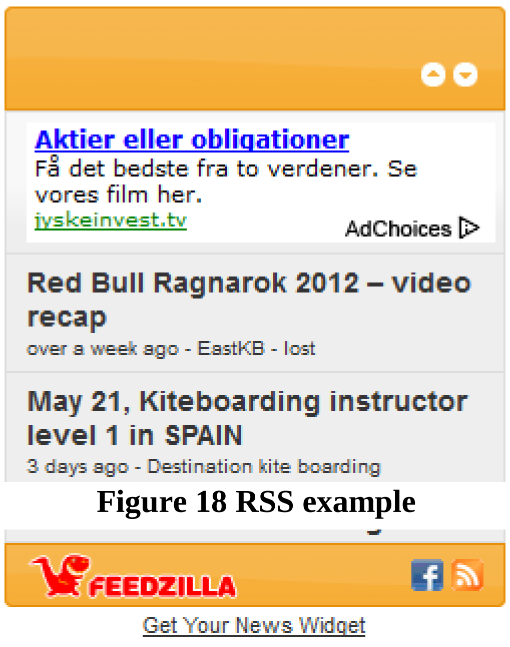
2.10.1. Analysis

Our goal was to create an interface where the user would easily read different news from the internet. The way we decided to approach this was by using RSS feeds. RSS stands for Really Simple Syndication and is a family of web feed formats used to publish frequently updated works—such as in our case News Feeds.

Version 1

2.10.2. Design

After "google-ing" we found a small tool provided by the friendly people at FeedZila that was generating a widget (Java Script + Html code[8]). This code we saved in an html file and we were passing the path of the file to the frame in our WPF User Control. This was the first version for the news (Vers.1) an easy solution but it wasn't providing us with a lot of control over the widget.



Aktier eller obligationer
Få det bedste fra to verdener. Se vores film her.
jyskeinvest.tv
AdChoices

Red Bull Ragnarok 2012 – video recap
over a week ago - EastKB - lost

May 21, Kiteboarding instructor level 1 in SPAIN
3 days ago - Destination kite boarding

FEEDZILLA
Get Your News Widget

Figure 18 RSS example

Therefore we figured we could use the XmlDataProvider to load and parse the XML feed, back from the given RSS link. This way we were binding the UI elements to the static resources provided by the XmlDataProvider. In order to display the items we were using a DataTemplate. The binding does not have to be done at the grid level. We could bind the ListItem and other controls to the data within the XmlDataProvider individually. However, there is a synchronization issue when multiple controls bind directly to the XmlDataProvider and use the XPath statement, therefore the use of IsSynchronizedWithCurrentItem property. All lists and arrays in .net implement the IEnumerable which makes data access to a specific position way easier. Since both

```
<Grid.DataContext>
    <XmlDataProvider x:Name="rssFeed" Source="url" XPath="rss/" />
</Grid.DataContext>
<DataTemplate x:Key="ItemTemplate">
    <Label Content="{Binding XPath=Title}"/>
</DataTemplate>
<ListBox x:Name="ListBox" Margin="0,0,0,20" DockPanel.Dock="Left"
    ItemsSource="{Binding}"
    ItemTemplate="{StaticResource ItemTemplate}"
    IsSynchronizedWithCurrentItem="True" />
<Frame Source="{Binding XPath=Link}" />
```

XmlDataProvider and ListBox are IEnumerable we don't have to worry about telling the compiler which RSS item to put on which list position. And as soon as the ListBox changes the SelectedItem the rest of the elements bound to the rssFeed will be notify automatically.

Version 2

But this was not enough because we wanted to have more RSS links available for the user to select from. And since the XmlDataProvider isn't giving us a dependency property we could change the RSS link given to the XmlDataProvider since it was hardcoded inside XAML. Thus we decided to make our own classes to hold the RSS feed. One of them is the RssFeedItem which is our base template for all the feeds. It contains 4 fields (Properties) Author, Title, Description and Link. These are the standard attributes in RSS 1.0. We wanted to also use the pubDate so the user would know when the respective news was released. Here came the problem because not all of the feeds had the pubDate element therefore we were getting exceptions. We did this in order to have a common base with all the RSS Feeds out there (since later on we would like to also enable the user to add his own RSS links)

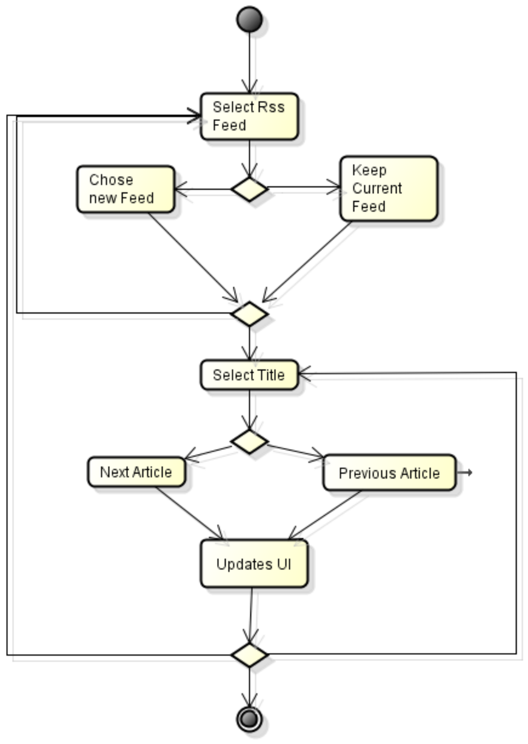


Figure 19 Activity diagram RSS feeds

2.10.3. Implementation

RssHandler parses thru the xml provided as an URL to the readFeed() method and returns a list of RssFeedItem. This is done simply by creating a WebRequest to the specific URL and then use a DataSet to parse thru the stream returned from the URL like so.

```
 List<RssFeedItem> rssFeedItems = new List<RssFeedItem>();
HttpWebRequest rssFeed = (HttpWebRequest)WebRequest.Create(url);
using (DataSet rssData = new DataSet())
{
    rssData.ReadXml(rssFeed.GetResponse().GetResponseStream());
    foreach (DataRow dataRow in rssData.Tables["item"].Rows)
    { rssFeedItems.Add(new RssFeedItem(...));}

}
```

Since we were going to have more RSS links available for the user (we ended up with 5) we made the RssNewsFeeds class which holds a list of RssFeedItem and the RSS URL (link). When this class is instantiated with a correct RSS URL (link) it calls the RssHandler to parse the given link and stores it inside the rssFeed the list returned by the handler. This way for each link we have an object of RssNewsFeeds with the corresponding list of feed items. Now that we had our data objects we needed to set them in a model and bind it to our view.

In the beginning of the project we decided we would use the MVC pattern (Model-View-Controller) because we were already familiar with it from earlier projects but after researching we found out that the MVC pattern is obsolete in WPF and that the way to go is MVVM (Model-View-ViewModel). The MVVM pattern works in such a way that it makes everything easy for the user. So we have the View (News.xaml) which represents our User Interface and it keeps in sync with the data from the model by binding it. The binding is done by adding the instance of our Model (in our case NewsViewModel) to the Application (App) Resources and then use it in our DataContext. The reason we add it at startup is that we want to load the rssFeeds(getting the feeds from the net takes a bit of time).

```
 Resources.Add("NewsViewModel", NewsViewModel.Instance);

 <Grid x:Name="main" DataContext="{StaticResource NewsViewModel}">
```

Another key element here is inside the ViewModel and that is the implementation of the NotifyPropertyChanged event handler. Whenever a property inside the Model class we notify by issuing an event to the ViewModel with the specific property name and the viewModel in turn will notify the View to display the current value.

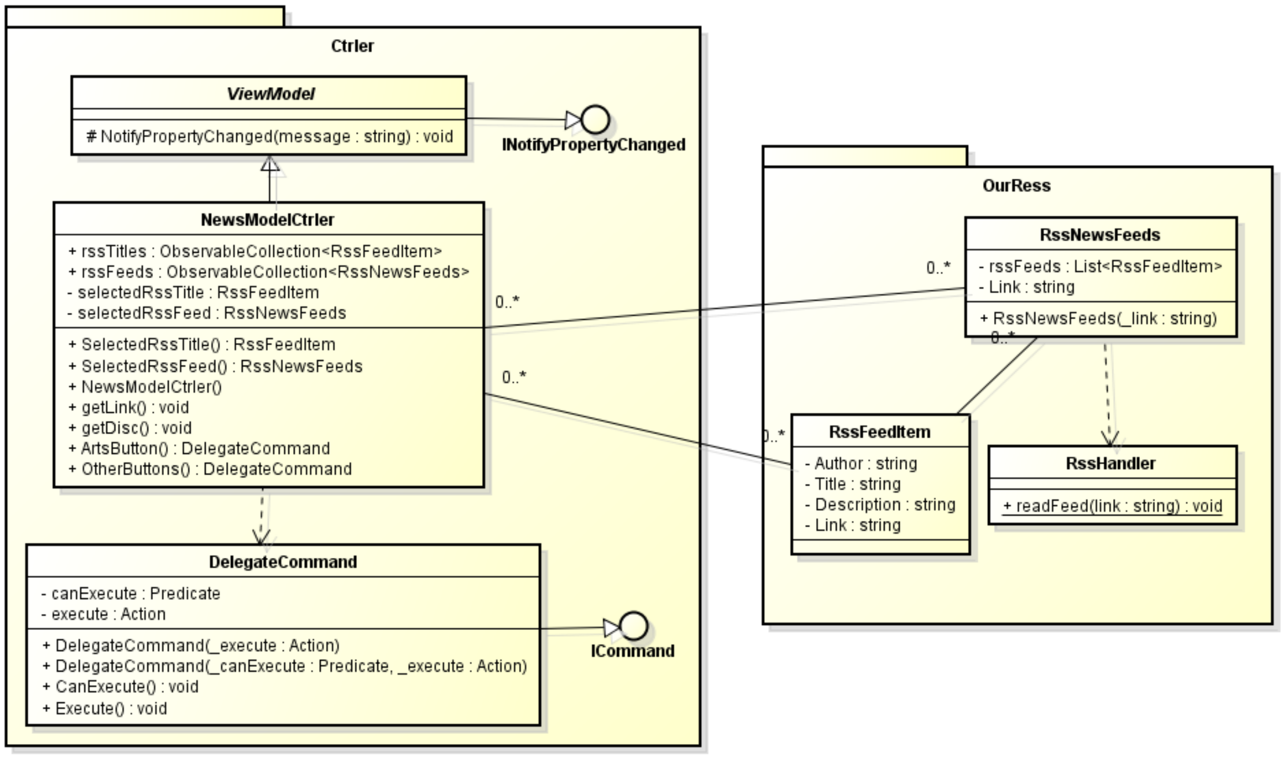
```
     public abstract class ViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected void NotifyPropertyChanged(string propertyName)
        {
            if (this.PropertyChanged != null) this.PropertyChanged(this,new
PropertyChangedEventArgs(propertyName));
        }
    }
```

The model part consists of NewsViewModel which holds one List (rssFeeds) and one ObservableCollections (rssTitles). The ObservableCollections class exists in the .net API and is specially made for binding to UI Elements which can hold more items such as ListBoxes and ComboBoxes. As soon as any changes are made on the list the UI will be notified and will display the corresponding items.

Initially we were holding the RSS Links inside a comboBox(binding the comboBox.ItemSource to our rssFeeds) and when the selectedItem(object of type RssNewsFeeds) inside the comboBox would change the property SelectedRssFeed on the model would be notify() clearing the rssTitles and adding the new list of RssFeedItems from the sent value(selectedItem inside the comboBox) also sets the SelectedRssTitle to the first index of the list. Same idea with the ListBox when the selectedItem changes the SelectedRssTitle gets notified and sets the new selectedRssTitle and notifies in turn the properties bound to the TextBlock and the Frame. A thing to mention here when the above happens is the UI calls the set{} of SelectedRssTitle.

In the TextBlock and Frame cases they only have get{} fields because we don't expect the UI to change them just to show when they get changed from behind. Later on when we got to playing with the switching focus between the controls (buttons mostly) and creating click events, the comboBox was hard to manipulate so we simplified it by adding 5 buttons. On each button click we switch to the respective list index by setting the SelectedRssFeed property and the feed is changed. So we have Arts(0),Computers(1),Consumer(2),Defense(3),Politics(4).

Since we are avoiding using event handlers behind the code, we can use the RoutedEvents provided by the DataContext binding. This way we can bind a Command to a Property that implements the ICommand. After researching the topic we found out that there are many stages in the execution of a command such as (save before execution, load after execution) also the ones we care about CanExecute and Execute. CanExecute checks to see if the command should be executed and if so than sets the flag to true. CanExecute is a Boolean function which takes no parameters (Predicate) and Execute is just a normal function (no return or parameters)(Action). Therefore we decided to make a "generic" class for our commands. _canExecute is defined as a Predicate and _execute is defined as an Action. This way we can define custom commands for each object of DelegateCommand. When this class is initialized the SelectedRssFeed is initialized to the first index of the rssFeeds.



Ctrler
ViewModel
# NotifyPropertyChanged(message : string) : void
INotifyPropertyChanged
NewsModelCtrler
+ rssTitles : ObservableCollection<RssFeedItem>
+ rssFeeds : ObservableCollection<RssNewsFeeds>
- selectedRssTitle : RssFeedItem
- selectedRssFeed : RssNewsFeeds
+ SelectedRssTitle() : RssFeedItem
+ SelectedRssFeed() : RssNewsFeeds
+ NewsModelCtrler()
+ getLink() : void
+ getDisc() : void
+ ArtsButton() : DelegateCommand
+ OtherButtons() : DelegateCommand
DelegateCommand
- canExecute : Predicate
- execute : Action
+ DelegateCommand(_execute : Action)
+ DelegateCommand(_canExecute : Predicate, _execute : Action)
+ CanExecute() : void
+ Execute() : void
ICommand
0..*
0..*
OurRess
RssNewsFeeds
- rssFeeds : List<RssFeedItem>
- Link : string
+ RssNewsFeeds(_link : string)
0..*
0..*
0..*
RssFeedItem
- Author : string
- Title : string
- Description : string
- Link : string
RssHandler
+ readFeed(link : string) : void

Figure 20 Detail of the class diagram

UI Binding: `<Button Command="ConsumerButton" />`

Handler Implementation:

```
   public DelegateCommand ConsumerButton
        { get
            { return new DelegateCommand((p) =>
                { if (isConn)
                        SelectedRssFeed = rssFeeds.ElementAt(2);
                });
            }        }
```

But again the pressing of buttons with the headset controls came in the way so we reverted back to Click events inside of which we get the command from the bound Model inside the DataContext and lastly execute it.

```
model = (NewsViewModel)Resources["NewsViewModel"];
private void btn5_Click(object sender, RoutedEventArgs e)
        {

            ICommand myComm = model.ConsumerButton;
            myComm.Execute("ConsumerButton");
        }
```

Scrolling through the list of tittles is done by 2 buttons which increase or decrease the index and focuses the new selected item and automatically notify the model. In the code behind the UI, besides the timers also present in the rest of our user controls,(more detailed explained in the UI section) there is an additional one designed to simulate a Key Down event on the frame each 5 seconds so the user can read the webpage. This proved more difficult to implement than initially anticipated because none of the approaches we took seemed to pay off. We tried a few things that in theory should've worked but with no results. The most one common around different forums was this one which would raise an event in which we define (the type, on what element to raise it on, at what time, what event) on the current user control.

```
this.RaiseEvent(new KeyEventArgs(Keyboard.PrimaryDevice,
PresentationSource.FromVisual(browser), 0, Key.Down)
                            { RoutedEvent = UIElement.KeyDownEvent });
```

Eventually we decided to use a small project we found on the internet called InputSimulator [9]. What this class does is that it raises Global Windows key events. What we mean by Global is that the event will happen on the selected UI inside Windows. So since the user won't use a mouse and won't go out of the application borders the event will also maintain only inside of the application (since from start it would never lose focus). It's not the best way to do things and we would want to improve on this particular feature.

2.10.4. Test

Since for the news we chose to use the MVVM pattern for implementing and updating the News tab it was easier to test the functionality thru Unit Testing. This was because we just had to test the functionality of the NewsViewModel class. In order to Unit Test we used Visual Studio to generate all the test methods for the selected classes. The implementation for each method was similar. We knew what each method was supposed to do/return so after calling the method we were checking if what we got was what we expected. This test method actually tests to see when the class is initialized it actual sets the SelectedRssTitle to rssTitles[0].

```
[TestMethod]
        public void testRssTitle()
        {
            actual = NewsViewModel.Instance;
            RssFeedItem expected= actual.rssTitles.ElementAt(0);
            Assert.AreEqual(expected, actual.SelectedRssTitle,      "SelectedRssTitle
Property failed");
      }
```

Also for the button clicks we made separate function inside the NewsViewModel(can be seen in the final class diagram) to execute the commands and we check to see if the data changed accordingly. For example for ConsumerButton when we press it the selectedRssFeed needs to be rssFeeds[2].

```
[TestMethod]
        public void testConsumerButton()
        {
            actual.testConsumerButton();
            RssNewsFeeds expected = actual.rssFeeds.ElementAt(2);
            Assert.AreEqual(expected, actual.SelectedRssFeed, "ConsumerButton Failed");
        }
```

3. Results

Going through this project has been a journey for all of us. There have been ups and downs both in the development phases and the group dynamics. But we have learned a lot during the last five months and feel that we have grown but personally and professionally.

We set out to create a program to help the mobility disabled, and exactly that we have done. The main focus of our system was to create a system that could help the focus group communicate with the world and in that way give them some of their life back. We have done this to a very high level with the implementation of the mind keyboard or just Mboard which we have decided to call it.

Since the writing was the primary function our system we have had the greatest focus on this part and we have created a module that not only lives up to our original expectations but also that goes beyond. We haven't implemented an intelligent dictionary but we have word prediction and we have set the keyboard up so the most common letters comes first, and as a bonus we have added a text to speech function.

The second thing on our list of requirements was the home automation, this part would give the user the ability to control his or hers surroundings like adjusting the bed or turning the lights on or off. This part we have completed to a satisfactory level as well. We have a simple interface and we can control simple hardware, this of cause can be build to a bigger scale in future upgrades and there basically aren't any restrictions to what you can connect.

As an extra feature we decided to add some entertainment to the system, this wasn't in our original plan but we found out that adding a small game to the application wasn't a huge deal compared to what pleasure it could be to the user. The fact that we got this part done as well just proves to us, even more than before, that we truly have succeeded in this project.

The last feature that we added is the news section, this part was very important to us because we all enjoy keeping up with the world around us, and knows how important it is to keep up with the world. This again was successful simply because we managed to add that feature as well.

By looking through what we have done we must say that we have a pretty good result. And when you compare that with the fact that we have implemented a fully integrated control panel with no confusing extra features we can all only be proud of what we have managed to do here. We set out to make something that could really help people and even though there still are things that can be better we call it a huge success.

4. Discussion

Like we mentioned a bit before we mostly accomplished the things we set up to do and in some parts even more. But in some other parts there are still loads of room for improvement. One of the things which we still desperately want to improve is the writing. It was the main focus of our project and all the way thru we were thinking how we can make it even better. Using the .net Language Dictionary to give the user some suggestions while he's writing is only half of what we would want to do. The idea is to make the keyboard layout change depending on what is written (for example: text written so far "be" keyboard layout would show: "a, c, f, l, n, r, s, t, y"). The letters would be all the possibilities (in the dictionary) for the current text. This would increase writing speed a lot since we wouldn’t have to iterate thru the entire alphabet letters every time but only the brief selection.

For the automation it would depend from user to user what they would like to hook up to our interface. The only thing we could do is to maybe have a pack of small software for some of the customers’ requests. Things we can get out of the top of our heads would be light control or bed automation.

In the entertainment category we can definitely find more games to add for the user. Keep him entertained and happy. For the news it’s planed out for the next update to implement a way for the user to add his or her own RSS links not only the one hardcoded in the code. Furthermore we want to implement our custom browser that can be controlled by the headset. This helps the user get integrated into the community.

Overall, as like we said so many times already, we are quite satisfied with the product we assembled in the time we had. Like said a bit earlier there are lots of things we could add for the user to control with the headset. And bring even more dignity back to the users of the headset.

5. Conclusion

Connection & Event Handling

We managed to build our own “control panel” which connects to the headset and gets the data. After we understood their SDK it was easy to interpret and handle the events sent by the headset. The events we focused on were user added (Profiles), cognitive detection and training,

Designing the Graphical User Interface:

We think we ended up with an easy to use interface. For navigation purposes the use of timers seemed the best solution. For the layout of the keyboard we would say we made a practical by arranging the most used letters to be on the left most side. We also implemented different styles the user can choose from. Besides the style the user can also chose between English and Danish. At application closing time the settings will be exported to an xml.

Data Storage

We didn’t have after all that much data to deal with. Besides the user settings (language, style, game data) which we decided to store locally in a xml file. We had the profile data which we decided to store in the same directory as the control panel under binary files (the edk.dll takes care of writing the binary file).

6. References

[1] A. Cockburn, “Writing Effective Use Cases”, Addison-Wesley Longman, London, 2000, pp. 15-16.

[2] Use Case Diagram,http://www.tutorialspoint.com/uml/uml_use_case_diagram.htm, March 2012.

[3] Manual, “Emotiv Software Development Kit”, EmotivLtd, 2011, pp. 45 -55.

[4] Software architecture, http://msdn.microsoft.com/en-us/library/ee658098.aspx, Jun 2012.

[5] Introduction to WPF, http://msdn.microsoft.com/en-us/library/aa970268.aspx, Jun 2012.

[6] Frequency of the letters EN, http://oxforddictionaries.com/words/what-is-the-frequency-of-the-letters-of-the-alphabet-in-english , June 2012.

[7] Frequency of the letters DK, http://www.sttmedia.com/characterfrequency-danish, June 2012.

[8] Feedzila gadget generator http://widgets.feedzilla.com/news/builder/index.html June 2012.

[9] Input Simulator http://inputsimulator.codeplex.com/ Jan 2012

7. Appendices

7.1. Appendix A Project Description

7.1.1. Background Description

Although nowadays some would say that we are surrounded by technology like smart phones, TV's, and computers, and that this allows the users communicate easily. But there are still certain people/groups that don't have the opportunity to use and control them as we can. Physically impaired people cannot move any body part at all, but there brain is fully functional. Therefore, we would like to develop the "M-board" in cooperation with Horsens kommune that will engage those people with their environment more seamlessly and improve their lives, bring them closer to society.

The primary focus of our project is to build a digital spelling table (M-board) with a cursor hovering over the letters and along with the Emotiv EPOC device we will be able to detect specific impulses when the subject would want to chose a letter. This way they will be able to formulate words and even sentences, leading to their ability to communicate with others. The fact that the product requested from us is quite basic that doesn't mean we lack new ideas in which we would make it even better.

We think this is an important area to work in since we today have the technology to make the everyday life easier for a lot of people that are disabled in different ways. The reason for us to focus on this specific group is that the technology is cutting edge and we think it has the potential to become very big.

Even though today there already exists a lot of applications and instruments that help those people, for example: controlling PC using the eye or various intelligent devices. We would like to go more forward and to create the application that allows the user a control with only a thought. Just wonder, when the user will be able to control the house or other environment with his mind.

Seeing that this is a new technology it has a lot of room to evolve and the possibilities are endless. After we develop the basic components, we can build an endless number of software solutions.

The basic concept behind is to use the Emotive EPOC headset that can detect and process real time brain waves. It measures a wide range of thoughts and emotions and allows using brain activity in the application. The human brain is made up of billions of neurons. When we do various activates, neurons transmit messages with each other, and we can measure those signals with electrodes and amplify them and then later use them in our application, to the various actions we want to perform.

The Emotiv headset uses 14 sensors that measure the electric signals in our brain. They get readings in pairs and the difference in reading are the n treated as the signal. When the application will ask the certain action, it will compare how the values from the electrodes change. When the change will be similar, the application accomplishes the calling action. The Emotiv headset is able to recognize expressions, actions which transmit over Bluetooth and the Bluetooth dongle to the application that we can read. So, the Emotiv headset is kind of the interface between the user and computer to handle the user's actions.

Because the brain waves are relatively different for every user, we need to calibrate the application before use. The headset will read the users emotions, gestures, and the build in gyro and train actions for controlling movement in different directions. Then we can save these parameters for every user, and use them for controlling our “M-board”. Every user will have his own account with his own record data, so it can be applicable for many users.

As we mentioned at the beginning, we will mainly focus on physically impaired people, those who can’t communicate or express their needs. Therefore, the mind keyboard will be the main part of the program, which allows writing an email, communicating on the social networks or doing a small work on the PC. Not only that the application will handle many functionality such as the gaming, browsing on the internet, listening to music, watching the movies or browsing pictures. It could be kind of library, with the simple interface going through the favorite folders.

Although our software will not have using all of our ideas, however it will help and make life easier for many users. This is a great challenge for us to create the program for the people need to depend on others. On the other hand working with the Emotiv Headset will give us the opportunity to learn the latest technologies, that we think will be used more and more in the near future, in normal everyday life and in the gaming industry.

Specification of Purpose

The purpose of the project is to develop a mind – keyboard application (M-Board) for Horsens Kommune to help a person who has some condition that restricts their ability physically.

The basic idea is using emotive EPOC headset that feeds off the electroencephalograph (EEG) signals from the user’s brain and sent them to the mind-keyboard application that is able to translate them into commands. Physically impaired people will be engaged with their environment more seamlessly.

The basic part of our application allows its users to spell out the words that they want to use giving them the ability to talk not only to the persons around them but also across the internet. Also it will offer the user an entertaining system that he will be able to control. We will do this by hooking the headset up with a wireless connection to a laptop and then have a piece of software translate the signals into commands in our application.

Problem Formulation

As we went through the problem, we figured out, that we have basically two main parts of the application which we have to handle. Firstly we need to use the Emotiv Software Developer’s Kit to define a bridge between the Emotiv headset signals and our application to be able control various features in there. The second important part is to use suite sequences and implemented them with required functions.

Connection:

How to connect our application to the Emotiv Engine? How to get the source of all the signals in our application? Those questions are probably essential before we will be able to continue forward. We need to connect and build our own bridge that will connect the Emotiv device and our application. We also have to decide whether we are going to handle user profile in our application, or we will use the Emotive Control Panel. Maybe it will be better to have our own profile and just call user profiles from the Emotiv engine.

Structure of the events:

How to handle various events from the headset for more users? Because the headset generates various events (four categories) we need to handle those events and it might be good to create the data structure, where we can find out what event was generated according to the user's ID.

Selecting suite Commands:

We need to ask ourselves, what kind of functions we want to control? Which means that we have to consider how to control the navigation panel and the M-board? For example go down, up and enter functions. So we need to decide what sequences (signals) from the users would trigger those functions. The Emotiv software provides us with 4 main categories of data: Expressive, Affective, Cognitive and Gyroscope data.

We need a control set that would work for people with a minimum amount of training. Although we haven't made a decision on what the commands to use, but we suggest that we need mental commands (affective) or facial gestures (Expressive) that will be fast to perform and easy to detected. But what if the user responds to certain feelings or mood, then he will not be able to write because of the smiling. Therefore we have to find out a suite of facial expression for the commands which can be used to perform actions and ignore all the other electric signals generated by muscle movement and still, easily use the small signals generated by the user. Maybe we should conside the gyroscope for some movements in the navigation panel.

Designing the Graphical User Interface:

How to design the GUI (navigation panel)? How to make the design of the keyboard and organize letters on the keyboard?

The GUI design must be as simple as possible to be easy to navigate through both when it comes to the keyboard and also through the navigation panel. The user should have feeling that automatic writing is done with the keyboard and computer. We also need to consider, how to design efficient mapping of the keys on the keyboard. We shouldn't use the common keyboard design, because the writing will be too slow for our application. There should be done a new algorithm.

Other questions: What size of the font we should use? How to change the type of keyboard from English to Danish and back?

Obviously, some of the people need to adjust size of the font when they are writing and also change settings such as the keyboard language, therefore we should consider it in our design, the suite place for the writing settings and easily manipulate with them.

Store Data:

How to save data? Should we store the data in xml files or a Database? Do we need to save any data?

That is a good question. Maybe we don't need to store any data, but we have to consider the best solution. We can call the information about the user profile, configurations and signals from the emotive control panel or build our own storage.

When we decided to build our own storage, then we have to consider the amount of data that we will be storing. Probably, there will be very small amount of data, therefore we would store data in an XML document and re-write the file any time any data changed. So it would be useful for configuration files and also for the user's profiles. We are expecting that the XML files never grew larger than 100kb.

7.1.2. Delimitations

Because the primary focus of our project is to help mobility disabled people be able to communicate with the people around, therefore the spelling words and writing the messages is mandatory part of the project and should be implemented. Although, we would like to implement more functionality to the application, probably we will not have enough time to do it. So, some of the features we will implement only if remains the time for it. Some will be quite easy to do because we will have the core of the system done so adding just a bit more functionality won't prove to be difficult such as:

- Browsing the internet
- Supporting keyboard languages
  (Mandatory support is EN and DK language)
- Watching movies
- Listening the music
- Browsing the pictures

But we also have other ideas that would make this software better for the customer such as:

- Gaming
- Supporting sound control
-

7.1.3. Choice of Model and Method Procedure

One of the most important aspects of developing any system is the methodology used, because they impose a disciplined process upon development more predictable and more efficient. There were many different models developed and debated, but only a fraction of them stood the test of time and proved useful. Motivated by the most popular trends in this field we choose to work with Agile Unified Process.
Agile Unified Process is a simplified version of Rational Unified Process, which takes the best practices of RUP and Agile Methodology. We redefine the phases of RUP to fit into Agile Unified Process [2].

- Inception
  - Project scope for our application
- Elaboration
  - Analyze our project in detail level. This phase is very important and at this level we should go through three kinds of models: Use Case Model, Domain Model, Interface Model
- Construction
  - Test and development are the major part of this phase
- Transition
  - On e of the major activities of this phase is to sign off the artifacts and deployment

In AUP an important perspective is that the Agile Unified Process is in small iterations. Every iteration will define the activity and then we can perform to build, validate and deliver working software.

As part of the AUP process we will use pair programming. We've already had experiences with that technique. Basically two programmers work together, one types code while the other is observer. Of course the roles can be switched. The main advantages of this methodology are that everyone knows what everyone is doing and they can be familiar with the whole system. This way, pair programming will enhance our team communication.

Next very well practices we have with refactoring methodology. Refactoring is a technique to restructure code in a disciplined way, a technique that is a fundamental practice of XP. The basic idea is that you make small changes to your code, called refactoring, to support new requirements and/or to keep your design as simple as possible. The advantage of refactoring is that it enables programmers to safely and easily evolve their code to fulfill new requirements or to improve its quality [1].

Another approach will be used what we call concurrent development. The principal of the methodology is that the some of the problems can be solve independently. So some of us will try to solve problem for a couple of hours independently and then discuss which came with the best and easier solution. This implements the most fitted solution in the project and will assure the best quality of the software.

At last for the testing process we would like to use the principles of Test Driven Development. This development combines to write unit tests first, before we write production code. We cannot write production code until you have written a failing unit test. The tests and code are written together, with the tests just a few seconds ahead of the code. The basic goal of TDD is to write clean code that works [3].

7.1.4. Time Schedule

Description	**Finished week**		
	Plan	**Forecast**	
		Week	**+/-**
Delivering project description	51	51	0
Making project plan – start project	3	2-3	-1
Making design architecture	4	4	0
Class definition complete, basic model between Emotiv API and application	5	5	0
GUI basic design complete , settings	6	5-6	-1
Basic core coding completed, recording signals	7	7-8	+1
Testing and debugging of everything completed and deployment	8	8	0
Basic model mind -keyboard , GUI design , algorithm	9	9	0
Basic core coding completed,	10 -11	11	+1
Testing and debugging of everything completed and deployment	12	12	0
Class definition complete, for more functionality	13	12-13	-1
Design GUI for other functionalities complete	13	14	+1
Application Build	14-16	16	1
Additional coding and testing completed and deployment	17-18	18	0
Documentation completed	22	22	-1

Final project report	23	23	0
Final deliveries	24	24	0

Activity Plan

Activity	**W3**	**W4**	**W5**	**W6**	**W7**	**W8**	**W9**	**W10-W11**	**W12**	**W13**	**W14-W17**	**W18**	**W19-W22**
Project plan	🟦												
System Design	🟦	🟦											
Program Design		🟦	🟦	🟦									
Implementation, signal pattern				🟦	🟦								
Test and Debugging, signal pattern					🟦	🟦							
GUI design, Keyboard design						🟦	🟦						
Implementation keyboard							🟦	🟦					
Test and Debugging								🟦	🟦				
Additional program design, GUI									🟦	🟦			
Additional implementation										🟦	🟦		
Testing											🟦	🟦	
Documenting	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

7.1.5. SOURCES, REFERENCES AND LITERATURE

Books:

[1] – Martin, Robert C, Clean Code, Apprentice Hall , ISBN 0-13-235088-2 , 2008

Articales:

http://www.nia.nih.gov/Alzheimers/Publications/Unraveling/Part1/neurons.htm , Neuroms and their Job

http://emotiv.com/ , homepage EMOTIV headset

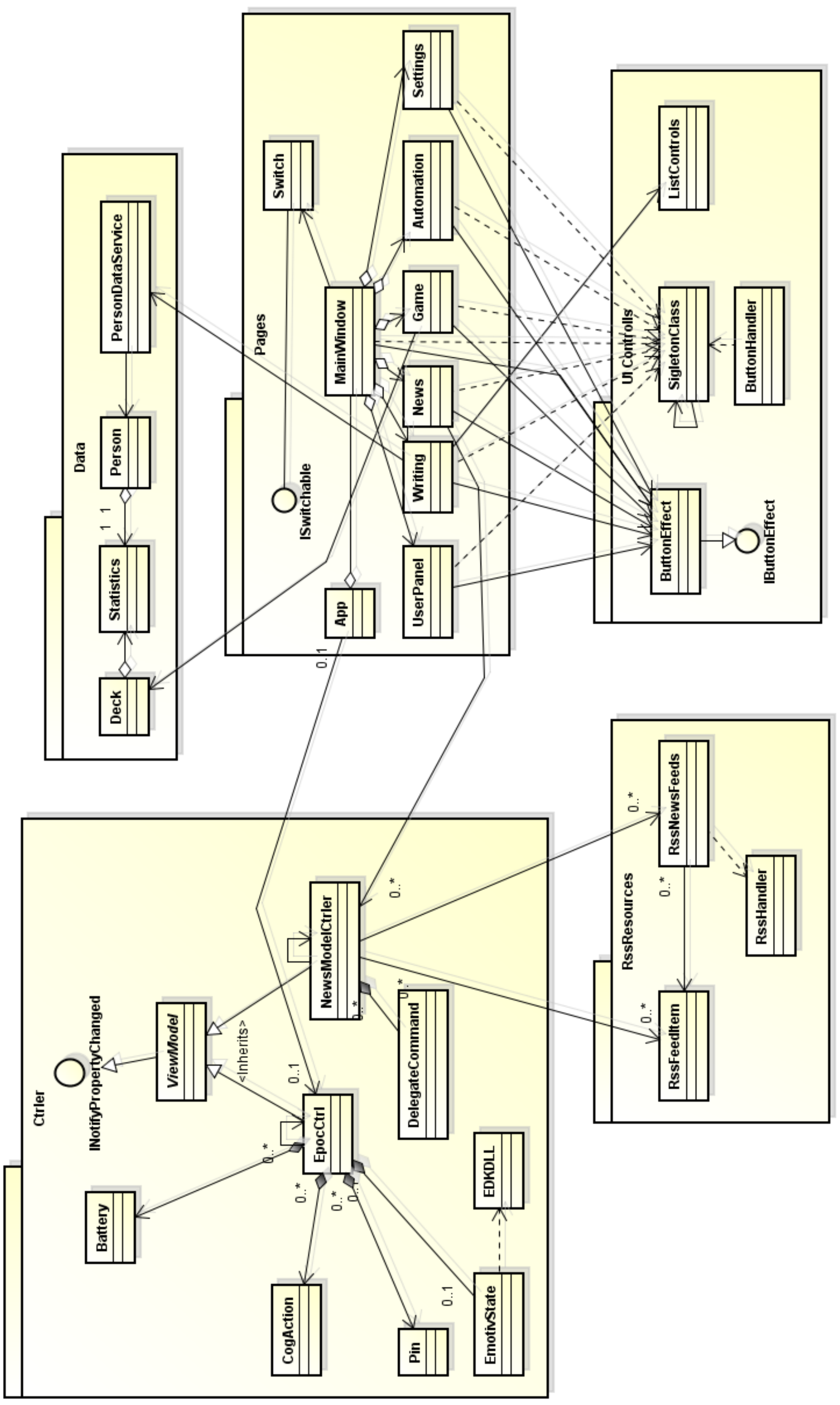
http://en.wikipedia.org/wiki/Emotiv_Systems , Wikipedia about Emotiv

[2] - http://www.ambysoft.com/unifiedprocess/agileUP.html , Agile Unified Process

[3] - http://www.agiledata.org/essays/tdd.html - TDD Test Driven Development.

7.2. Appendix B

7.2.1. Class Diagram



7.3. Appendix C

7.3.1. Use Case

USE CASE 1 Login to the Application

Brief Description: This use case describes how the use assistant login the user to the application.

Actor: An assister user

Flow of Events:

3. Assistant user selects the name of the user by clicking on the list box of the users and he will click on the button (Open profile)
4. The system logs in the user to the system, and actives his headset.

Pre-Conditions: The user has already created an account in the system and the headset is connected to the computer.

Post Condition:

Success Post-Condition: the user profile exists in the system and was activated his headset.

Failure Post-Condition: the system has a problem loading the user profile and after while will notice the assistant user to try again step 1.

USE CASE 2 Logout from the Application

Brief Description: This use case describes how the use assistant logout the user from the application.

Actor: An assister user

Flow of Events:

1. Assistant user clicks on the button on the button (Logout)
2. The system logs out the user from the system and will save actual setting to the binary file and xml file.

Pre-Conditions: The user has already been login in the system and the headset has still connected to the computer.

Post Condition:

Success Post-Condition: the user profile was logout from the system and saved to the external files.

Failure Post-Condition: the system has a problem to logout the user and will notice the problem.

USE CASE 3 Create a user profile

Brief Description: This use case describes how the use assistant creates the profile for the user.

Actor: An assister user

Flow of Events:

1. Assistant user inserts the name of the new user to the text box.
2. Then the assistant user clicks on the button create profile.
3. The system shows the confirmation window that the profile has been created.

4. The system logs in the new user.

Pre-Conditions: The headset is connected to the computer.
Post Condition:
Success Post-Condition: the user profile was created and the name is visible in the list of the users.
Failure Post-Condition: the system couldn’t create a new profile and will reject the message screen and notice the assistant user try it again.

USE CASE 4 Training user

Brief Description: This use case describes how the use assistant creates the profile for the user.
Actor: An assister user and user
Flow of Events:

1. Assistant user clicks on the list box and select the type of the action for the training.
2. Then the assistant user clicks on the button (training).
3. The system starts to read the signal from the users’ headset in 8 seconds..
4. Then system displays the confirmation screen and notices the user that training was completed.
5. The assistant user will select button (OK) on the confirmation screen or can process repeat from the point 2.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system.
Post Condition:
Success Post-Condition: the user is able to control application with the headset.
Failure Post-Condition: the user is not able control application.

USE CASE 5 Cancel Training

Brief Description: This use case describes how the use assistant can cancel training process.
Actor: An assister user and user
Flow of Events:

1. Assistant user clicks on the list box and select the type of the action for the training.
2. Then the assistant user clicks on the button (training).
3. The system starts to read the signal from the users’ headset in 8 seconds.
4. Then system displays the confirmation screen and notices the user to cancel the training process or save it.
5. The assistant user will select button (Cancel) on the confirmation screen.
6. The system will interrupt the training process and not save the last data from the training.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system.
Post Condition:

Success Post-Condition: the user will have previous training or none when training has been done for first time.
Failure Post-Condition: the user has loaded a new signal data from the last training.

USE CASE 6 Play Game

Brief Description: This use case describes how the use assistant and user can play a game.
Actor: An assister user and user
Flow of Events:
1. Assistant user clicks on the button (Game). The user will wait on focus button (Game) and in 3 seconds, will invoke event click button by the headset.
2. The system will load the game panel.
3. The assistant user can interrupt game anytime by clicking the other button from the main menu. The user must wait till the button (back to main) is focused and then invoke action by headset click.
4. The system will save the statistics from the game and return to the menu.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system.
Post Condition:
Success Post-Condition: the user and assistant user will load game panel and play game.
Failure Post-Condition: the game panel was not loaded and the message box is displayed.

USE CASE 7 Control Home Automation

Brief Description: This use case describes how the use assistant and user can control home automation.
Actor: An assister user and user
Flow of Events:
1. Assistant user clicks on the button (Home Automation). The user will wait on focus button (Home Automation) and in 3 seconds will invoke event click button by the headset.
2. The system will load the home automation panel.
3. The assistant user can click on the button up or down. The user must wait till the button (up or down) is focused and then invoke action by headset click.
4. The system will sent a signal out from the application and will control the home automation.
5. The assistant user can return back to the menu anytime by clicking the other button from the main menu. The user must wait till the button (back to main) is focused and then invoke action by headset click.
6. The system will return to the main menu.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system.

Post Condition:

Success Post-Condition: the user and assistant user can control home automation.

Failure Post-Condition: the home automation is not available and the message box is displayed.

USE CASE 8 Writing Text

Brief Description: This use case describes how the use assistant and user can write text in the writing panel.

Actor: An assister user and user

Flow of Events:

1. Assistant user clicks on the button (Writing). The user will wait on focus button (Writing) and in 3 seconds will invoke event click button by the headset.
2. The system will load the Writing panel.
3. The assistant user can click on the buttons (keys). The user must wait till the buttons (keys) are focused and then invoke action by headset click.
4. The system will write selected letter to the text field in the panel.
5. The assistant user can return back to the menu anytime by clicking the other button from the main menu. The user must wait till the button (back to main) is focused and then invoke action by headset click.
6. The system will return to the main menu.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.

The user is a log in to the system.

Post Condition:

Success Post-Condition: the user and assistant user were able to write text to the text field.

Failure Post-Condition: the writing panel was not available and the message box is displayed.

USE CASE 9 Reading News

Brief Description: This use case describes how the use assistant and user can read news in the application.

Actor: An assister user and user

Flow of Events:

1. Assistant user clicks on the button (News). The user will wait on focus button (News) and in 3 seconds will invoke event click button by the headset.
2. The system will load the News panel with the RSS frame.
3. The assistant user can click on the buttons (up and down). The user must wait till the buttons (up and downs) are focused and then invoke action by headset click.
4. The system will load selected news from the list box to the frame.

5. The assistant user can return back to the menu anytime by clicking the other button from the main menu. The user must wait till the button (back to main) is focused and then invoke action by headset click.
6. The system will return to the main menu.

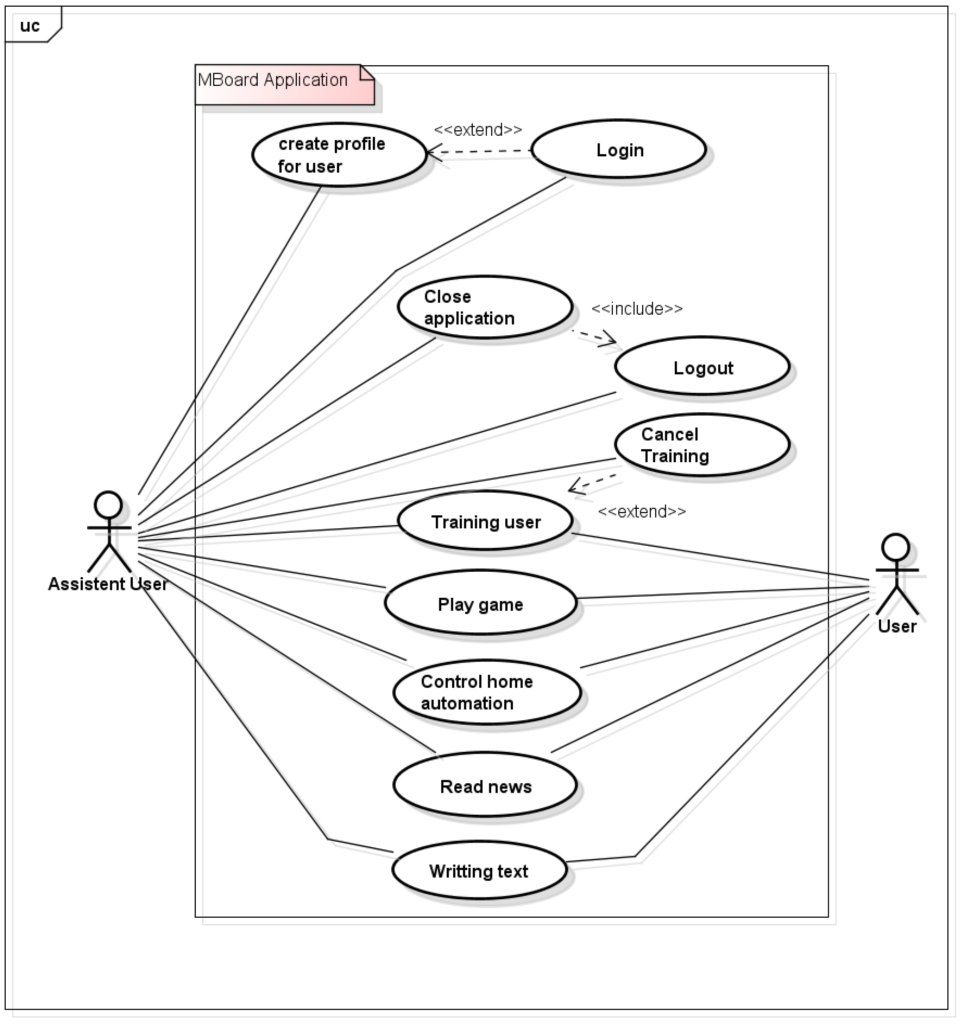
Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system. The computer is connected to the Internet network.
Post Condition:
Success Post-Condition: the user and assistant user can read the news in the frame.
Failure Post-Condition: the news is not loaded to the frame or the computer doesn't have connection to the Internet network and message box is displayed.

USE CASE 10 Close Application

Brief Description: This use case describes how an assister user can close the application.
Actor: An assister user
Flow of Events:
1. Assistant user clicks on the button (Close).
2. The system will save the user setting to the xml file and logs out the user from the application.
3. The system is closed.

Alternative flow of Events:
1. Assistant user clicks on the button (Close).
2. The system will save settings and close the application.

Pre-Conditions: The headset is connected to the computer and the user is a log in to the system.
Post Condition:
Success Post-Condition: the user's setting are saved and application is closed.
Failure Post-Condition: the message error is displayed.



Use Case Diagram

7.4. Appendix D

7.4.1. System Test Plan

Test CASE 1 Login to the Application

Brief Description: This test case checks how the use assistant login the user to the application.
Scenario:

5. Assistant user selects the name of the user by clicking on the list box of the users and he will click on the button (Open profile)
6. The system logs in the user to the system, and actives his headset.

Pre-Conditions: The user has already created an account in the system and the headset is connected to the computer.
Post Condition:
Success Post-Condition: the user profile exists in the system and was activated his headset.
Failure Post-Condition: the system has a problem loading the user profile and after while will notice the assistant user to try again step 1.
Expected Result:
The user can log in to the system and load a profile
Actual Result:
User logged in and profile loaded

Test CASE 2 Logout from the Application

Brief Description: This test case checks how the use assistant logout the user from the application.
Scenario:

3. Assistant user clicks on the button on the button (Logout)
4. The system logs out the user from the system and will save actual setting to the binary file and xml file.

Pre-Conditions: The user has already been login in the system and the headset has still connected to the computer.
Post Condition:
Success Post-Condition: the user profile was logout from the system and saved to the external files.
Failure Post-Condition: the system has a problem to logout the user and will notice the problem.
Expected Result:
User will be logged out of the system
Actual Result:

Test CASE 3 Create a user profile

Brief Description: This test case checks how the use assistant creates the profile for the user.
Scenario:

5. Assistant user inserts the name of the new user to the text box.
6. Then the assistant user clicks on the button create profile.
7. The system shows the confirmation window that the profile has been created.
8. The system logs in the new user.

Pre-Conditions: The headset is connected to the computer.
Post Condition:
Success Post-Condition: the user profile was created and the name is visible in the list of the users.
Failure Post-Condition: the system couldn't create a new profile and will reject the message screen and notice the assistant user try it again.
Expected Result:
User profile will be created
Actual Result:
Profile created and saved to hdd.

Test CASE 4 Training user

Brief Description: This test case describes checks the use assistant creates the profile for the user.
Scenario:
6. Assistant user clicks on the list box and select the type of the action for the training.
7. Then the assistant user clicks on the button (training).
8. The system starts to read the signal from the users' headset in 8 seconds..
9. Then system displays the confirmation screen and notices the user that training was completed.
10. The assistant user will select button (OK) on the confirmation screen or can process repeat from the point 2.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system.
Post Condition:
Success Post-Condition: the user is able to control application with the headset.
Failure Post-Condition: the user is not able control application.
Expected Result:
User trains and profile get updated
Actual Result:
Training successful for both neutral and push

Test CASE 5 Cancel Training

Brief Description: This test case checks how the use assistant can cancel training process.
Scenario:
7. Assistant user clicks on the list box and select the type of the action for the training.
8. Then the assistant user clicks on the button (training).
9. The system starts to read the signal from the users' headset in 8 seconds.
10. Then system displays the confirmation screen and notices the user to cancel the training process or save it.
11. The assistant user will select button (Cancel) on the confirmation screen.
12. The system will interrupt the training process and not save the last data from the training.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system.
Post Condition:
Success Post-Condition: the user will have previous training or none when training has been done for first time.
Failure Post-Condition: the user has loaded a new signal data from the last training.
Expected Result:
Training gets canceled
Actual Result:
Training data is discarded

Test CASE 6 Play Game

Brief Description: This test case checks how the use assistant and user can play a game.
Scenario:

5. Assistant user clicks on the button (Game). The user will wait on focus button (Game) and in 3 seconds, will invoke event click button by the headset.
6. The system will load the game panel.
7. The assistant user can interrupt game anytime by clicking the other button from the main menu. The user must wait till the button (back to main) is focused and then invoke action by headset click.
8. The system will save the statistics from the game and return to the menu.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system.
Post Condition:
Success Post-Condition: the user and assistant user will load game panel and play game.
Failure Post-Condition: the game panel was not loaded and the message box is displayed.
Expected Result:
User can play a game and a user will be found
Actual Result:
Game played and score updated
Test CASE 7 Control Home Automation

Brief Description: This test case checks how the user assistant and user can control home automation.
Scenario:

7. Assistant user clicks on the button (Home Automation). The user will wait on focus button (Home Automation) and in 3 seconds will invoke event click button by the headset.
8. The system will load the home automation panel.
9. The assistant user can click on the button up or down. The user must wait till the button (up or down) is focused and then invoke action by headset click.
10. The system will sent a signal out from the application and will control the home automation.

11. The assistant user can return back to the menu anytime by clicking the other button from the main menu. The user must wait till the button (back to main) is focused and then invoke action by headset click.
12. The system will return to the main menu.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system.
Post Condition:
Success Post-Condition: the user and assistant user can control home automation.
Failure Post-Condition: the home automation is not available and the message box is displayed.
Expected Result:
Output will come from the ports on the rs232 board
Actual Result:
Output raises to 5 V just as predicted.

Test CASE 8 Writing Text

Brief Description: This test case checks how the user assistant and user can write text in the writing panel.
Scenario:

7. Assistant user clicks on the button (Writing). The user will wait on focus button (Writing) and in 3 seconds will invoke event click button by the headset.
8. The system will load the Writing panel.
9. The assistant user can click on the buttons (keys). The user must wait till the buttons (keys) are focused and then invoke action by headset click.
10. The system will write selected letter to the text field in the panel.
11. The assistant user can return back to the menu anytime by clicking the other button from the main menu. The user must wait till the button (back to main) is focused and then invoke action by headset click.
12. The system will return to the main menu.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system.
Post Condition:
Success Post-Condition: the user and assistant user were able to write text to the text field.
Failure Post-Condition: the writing panel was not available and the message box is displayed.
Expected Result:
The user can write text
Actual Result:
Writing works fine as expected and even more the system can read what you just wrote

Test CASE 9 Reading News

Brief Description: This test case checks how the user assistant and user can read news in the application.

Scenario:

7. Assistant user clicks on the button (News). The user will wait on focus button (News) and in 3 seconds will invoke event click button by the headset.
8. The system will load the News panel with the RSS frame.
9. The assistant user can click on the buttons (up and down). The user must wait till the buttons (up and downs) are focused and then invoke action by headset click.
10. The system will load selected news from the list box to the frame.
11. The assistant user can return back to the menu anytime by clicking the other button from the main menu. The user must wait till the button (back to main) is focused and then invoke action by headset click.
12. The system will return to the main menu.

Pre-Conditions: The headset is connected to the computer and has good signal with the all sensors.
The user is a log in to the system. The computer is connected to the Internet network.
Post Condition:
Success Post-Condition: the user and assistant user can read the news in the frame.
Failure Post-Condition: the news is not loaded to the frame or the computer doesn't have connection to the Internet network and message box is displayed.
Expected Result:
The user gets the news showed on the screen
Actual Result:
The news came up as it should
Test CASE 10 Close Application
Brief Description: This test case checks how an assistant user can close the application.
Scenario:

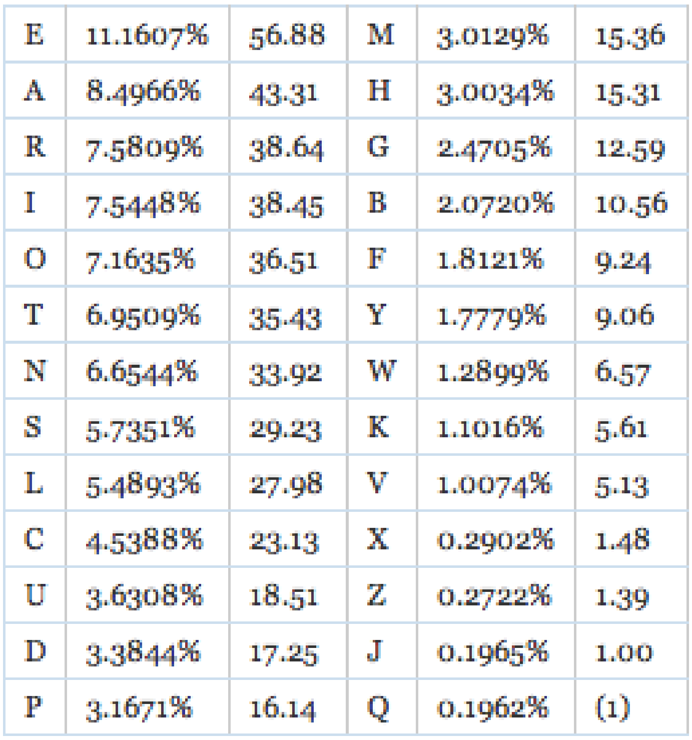
4. Assistant user clicks on the button (Close).
5. The system will save the user setting to the xml file and logs out the user from the application.
6. The system is closed.

Pre-Conditions: The headset is connected to the computer and the user is a log in to the system.
Post Condition:
Success Post-Condition: the user's settings are saved and application is closed.
Failure Post-Condition: the message error is displayed.
Expected Result: The System closes down and the profile is saved
Actual Result: Profile is saved and the application is terminated.

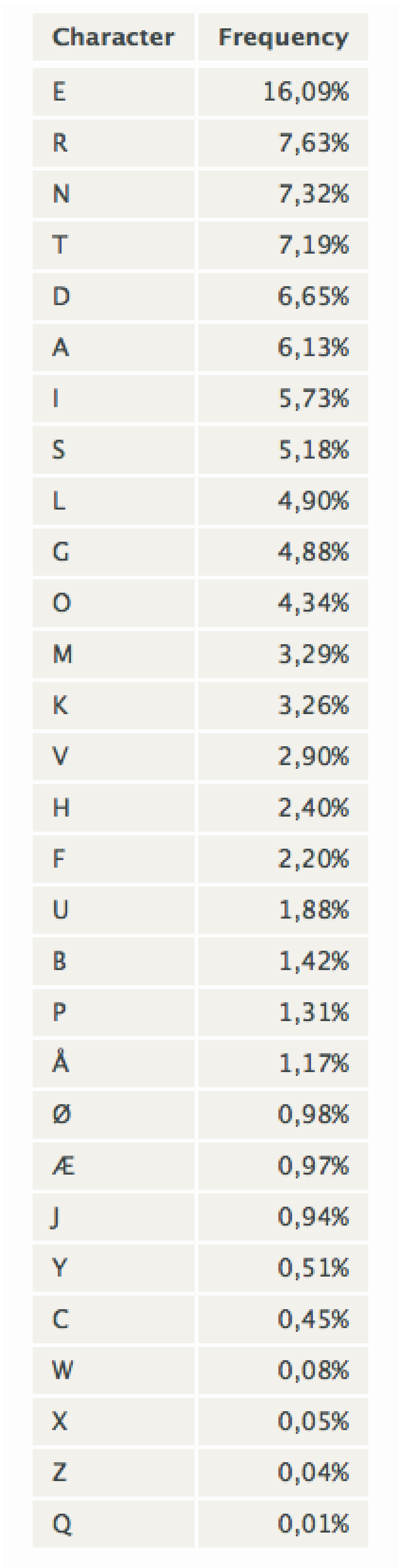
7.5. Appendix E

7.5.1. Frequency of letters

The frequency of the letters in English Text .[6]



E	11.1607%	56.88	M	3.0129%	15.36
A	8.4966%	43.31	H	3.0034%	15.31
R	7.5809%	38.64	G	2.4705%	12.59
I	7.5448%	38.45	B	2.0720%	10.56
O	7.1635%	36.51	F	1.8121%	9.24
T	6.9509%	35.43	Y	1.7779%	9.06
N	6.6544%	33.92	W	1.2899%	6.57
S	5.7351%	29.23	K	1.1016%	5.61
L	5.4893%	27.98	V	1.0074%	5.13
C	4.5388%	23.13	X	0.2902%	1.48
U	3.6308%	18.51	Z	0.2722%	1.39
D	3.3844%	17.25	J	0.1965%	1.00
P	3.1671%	16.14	Q	0.1962%	(1)



Character	Frequency
E	16,09%
R	7,63%
N	7,32%
T	7,19%
D	6,65%
A	6,13%
I	5,73%
S	5,18%
L	4,90%
G	4,88%
O	4,34%
M	3,29%
K	3,26%
V	2,90%
H	2,40%
F	2,20%
U	1,88%
B	1,42%
P	1,31%
Å	1,17%
Ø	0,98%
Æ	0,97%
J	0,94%
Y	0,51%
C	0,45%
W	0,08%
X	0,05%
Z	0,04%
Q	0,01%

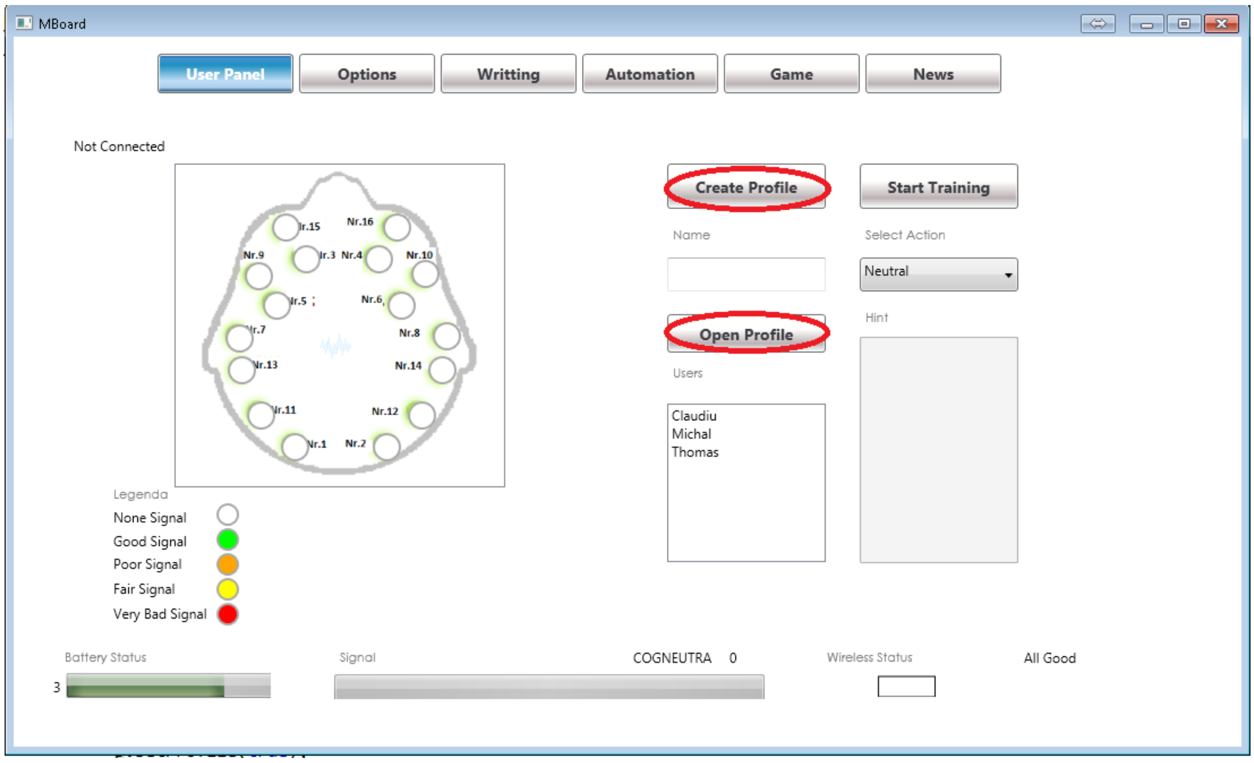
The frequency of the letters in Danish Text .[7]

7.6. Appendix F

7.6.1. User manual

User Panel

To create a Profile insert the User name into the Name field and press the Create Profile Button.
To open an existing profile, first select an Username from the Users Box and then click the Open Profile Button



Not Connected

Users: Claudiu, Michal, Thomas

Legenda	
None Signal	○
Good Signal	🟢
Poor Signal	🟠
Fair Signal	🟡
Very Bad Signal	🔴

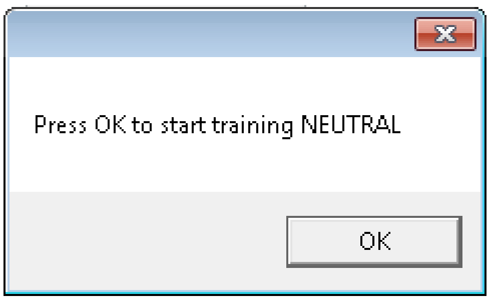
Battery Status 3

COGNEUTRA 0

All Good

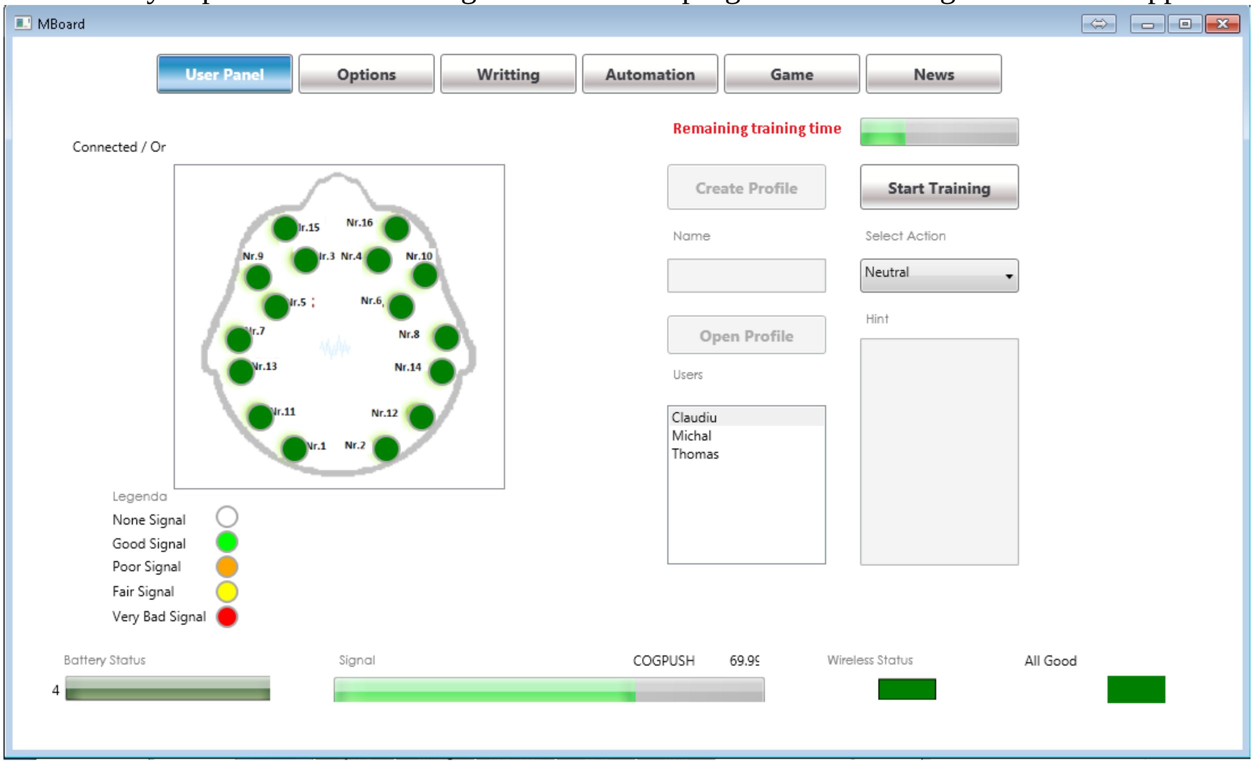
Whenever one of these buttons is clicked the application will go idle and start looking for the headset. In case it won't find the headset it will return an error message after 30 sec and you can retry reconnecting the Headset wireless dongle and try again. If the connection was successful the UI will update.

To start a training first chose the action to train after which press the Start Training button. After you press the button u will be notified with a pop-up message announcing you to get ready for the training. The training lasts for 8 seconds so concentrate.

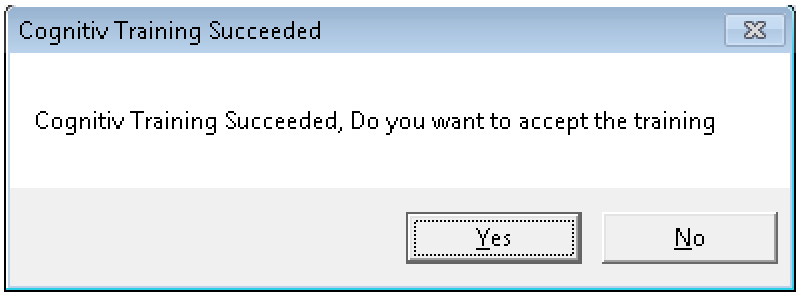


Press OK to start training NEUTRAL

As soon as you press OK the training will start and a progress bar showing the time will appear.



When the training will finish you will be asked if you want to accept the training. If you feel like you didn't concentrate long enough don't accept it.


Cognitiv Training Succeeded

Cognitiv Training Succeeded, Do you want to accept the training

When you are done training you can patiently for your selected button choice.

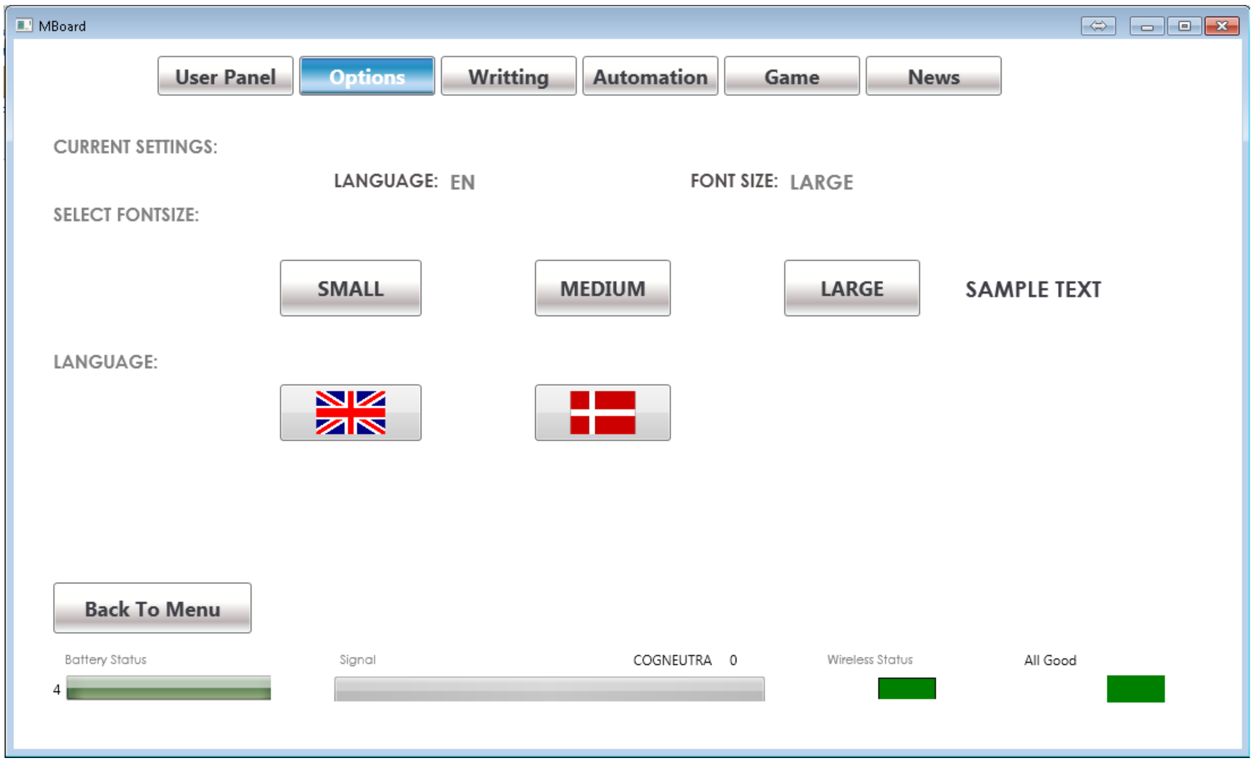
The control inside the application is done with the use of 3 small "helpers". As u can see in the picture above the User Panel is bluer than the rest of the buttons. We will call this from now on focus. This focus is one of the helpers. We'll call this the "Panel Focus". The second helper is another focus which resides in each page and it switches between the buttons inside the respective page. The third helper is the signal registered by the Application. If the signal is higher than 60 the focus will press the selected button. We will call this a "Click Event".

Each 3 seconds the button focus will move. When a "Click Event" is registered on one of the Panel Buttons the application will switch to the selected page. Once it's on the selected page the "Panel Focus" will stop moving and the "Page Focus" will start.

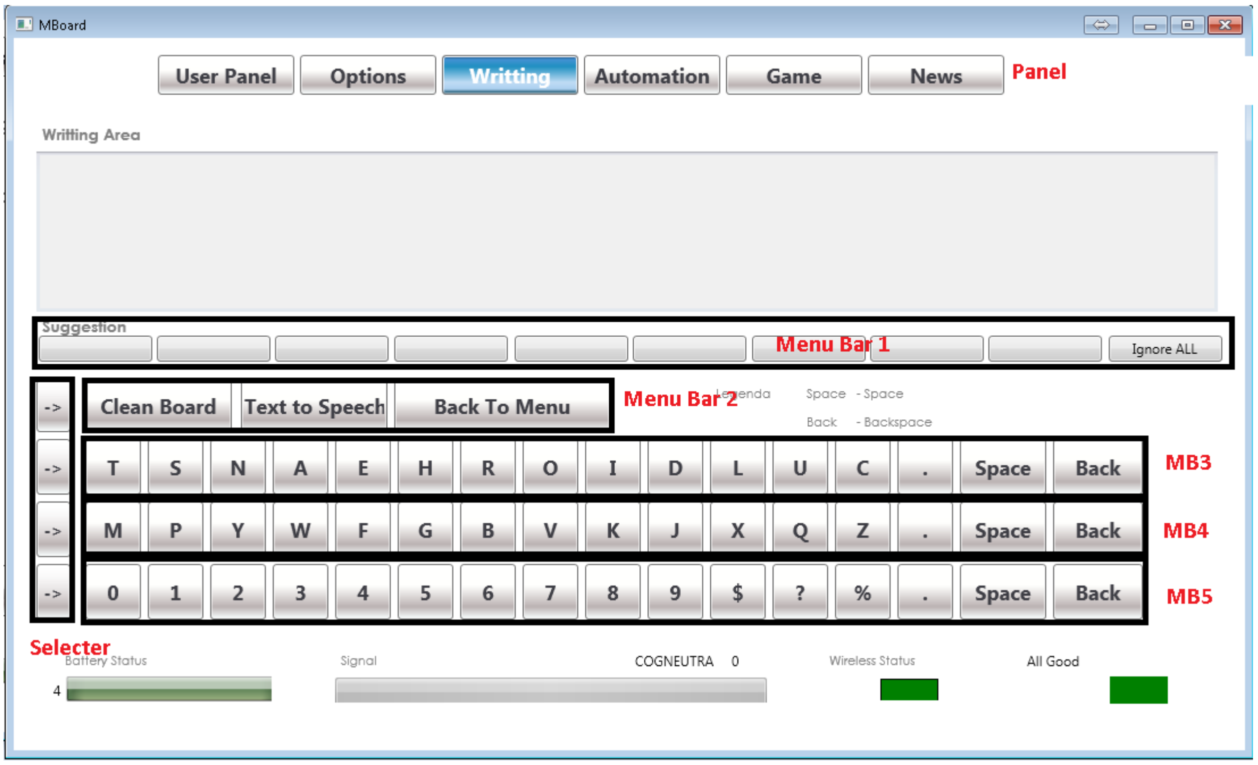
The Pages

Each of the pages has a "Back To Menu" button which will stop the "Page Focus" inside the current page and will restart the "Panel Focus" so you can chose another page.

The Option page



The Writing page



When you arrive on the writing page the first button in the Selector column will gain focus and each 3 seconds it will move down. When a “Click Event” will be registered the focus will start moving left on the selected Menu Bar (2, 3, 4, 5). In case you want another Menu bar you patiently wait till the focus moves until the last button (Back) and resets on the Selector. If the Engine will register a “Click Event” while the focus is on the Selector, the focus will start moving again between the Selector columns.

Menu Bar1

Suggestions: The application will be check the spelling and as soon as one of the typed words will be wrong, the focus will start moving between the Suggestions buttons.

Menu Bar 2

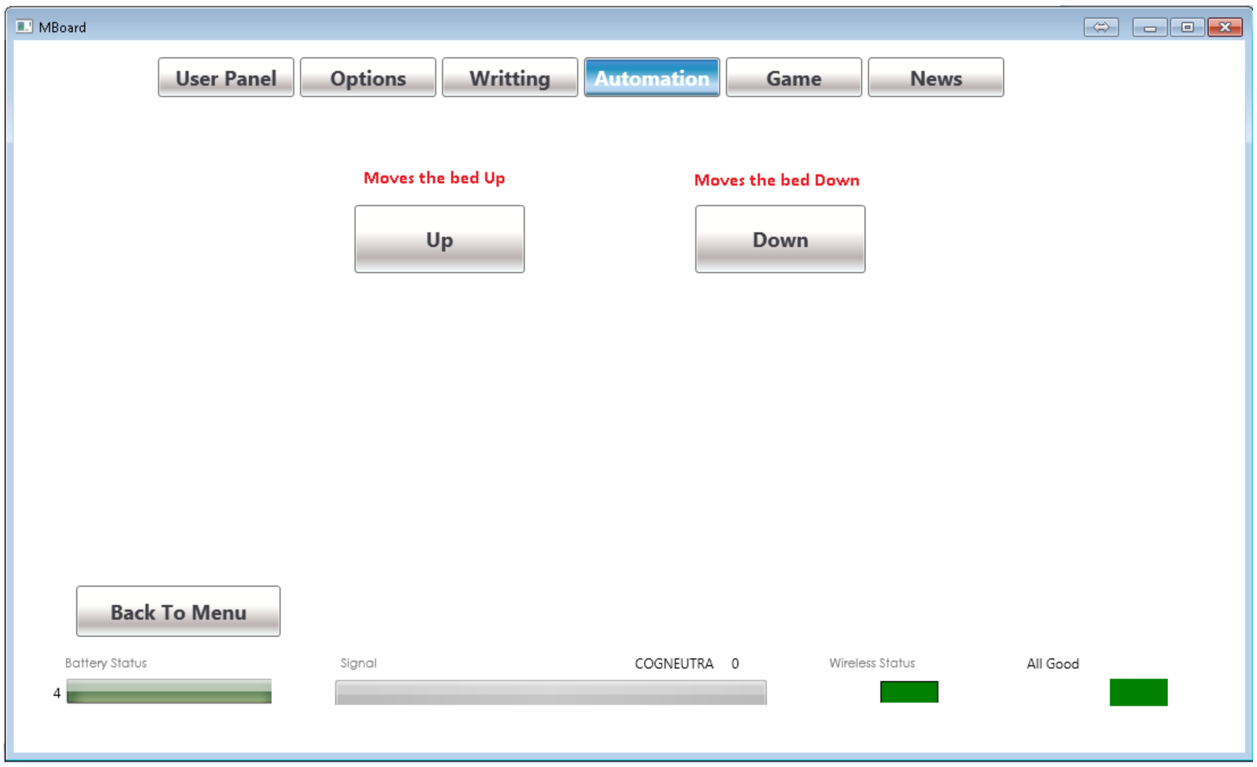
Clean Board: Clears the text in the Writing Area

Text to Speech: The application will read out loud the text inside the Writing Area (EN only)

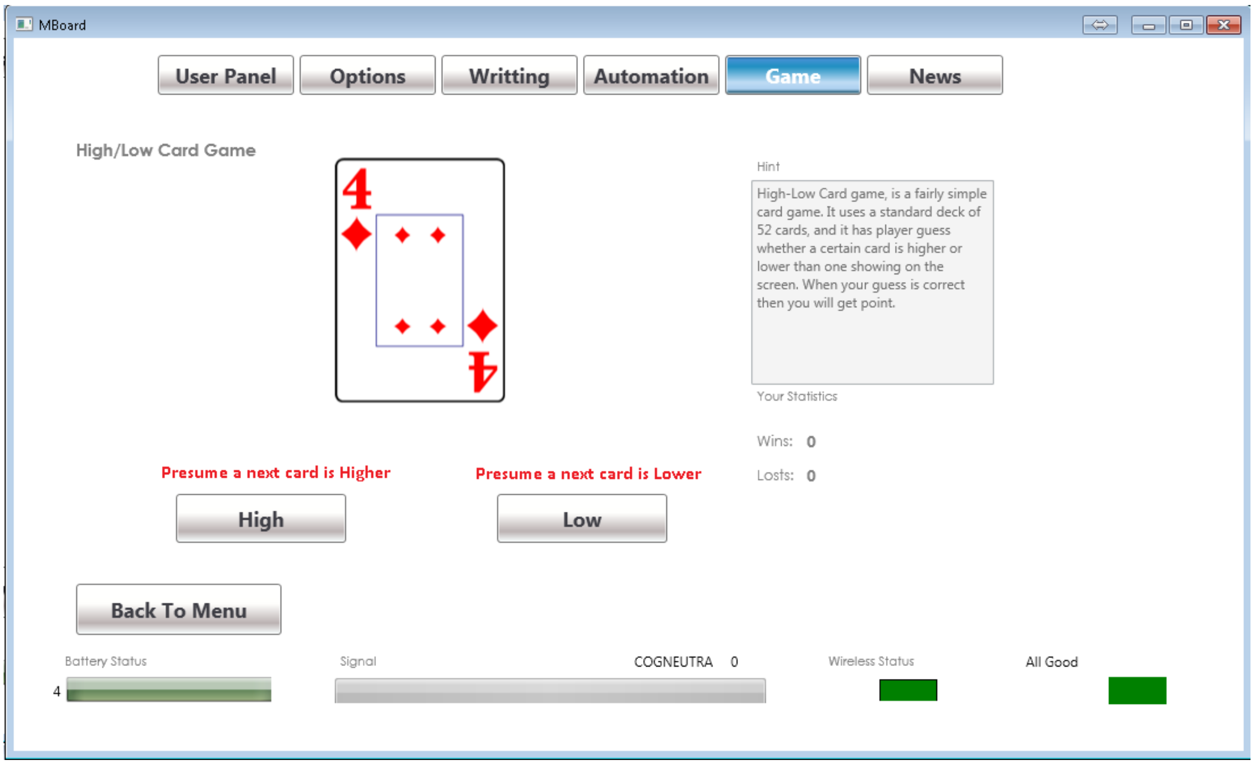
Back to Menu: The first button in the Panel will regain focus and you will be able to choice another Panel Button in order to switch to another page

Menu Bar 3, 4, 5

The Automation page

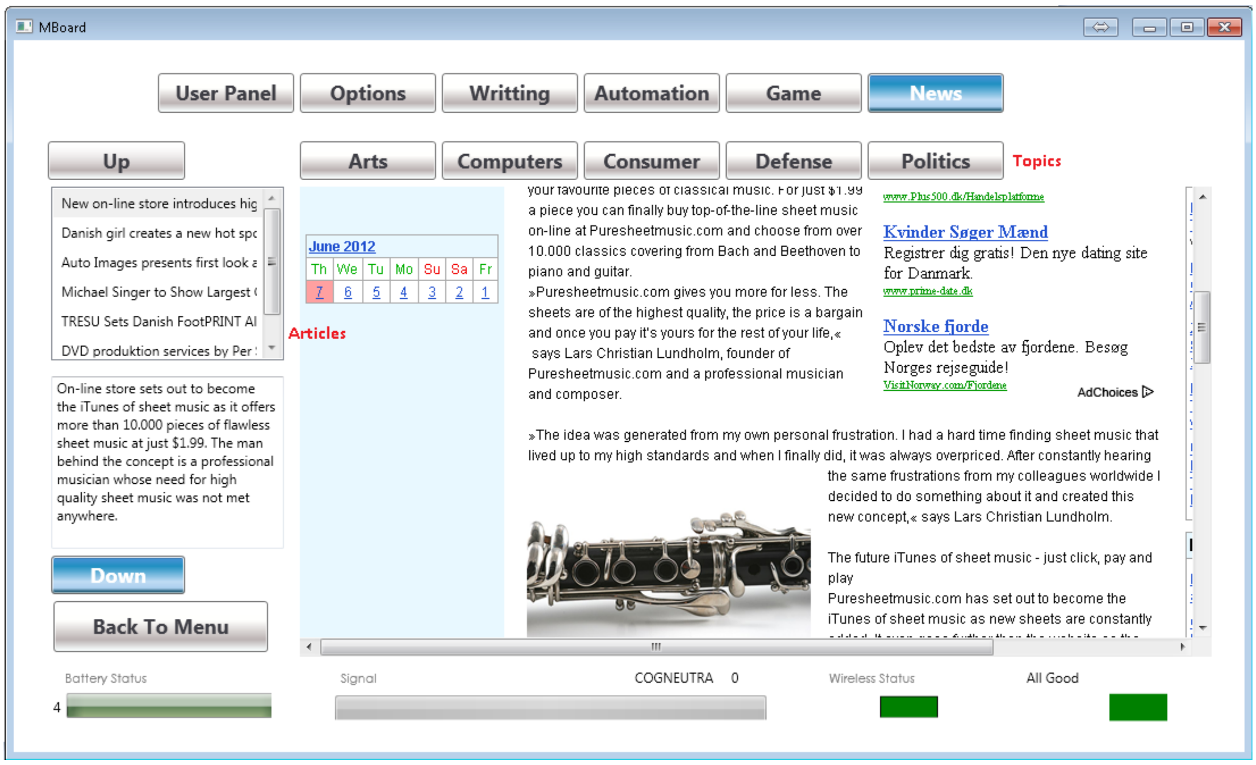


The Game Page



High-Low Card game, is a fairly simple card game. It uses a standard deck of 52 cards, and it has player guess whether a certain card is higher or lower than one showing on the screen. When your guess is correct then you will get point.

The News Page



New on-line store introduces hig
Danish girl creates a new hot spc
Auto Images presents first look a
Michael Singer to Show Largest (
TRESU Sets Danish FootPRINT AI
DVD produktion services by Per :

On-line store sets out to become the iTunes of sheet music as it offers more than 10.000 pieces of flawless sheet music at just $1.99. The man behind the concept is a professional musician whose need for high quality sheet music was not met anywhere.

your favourite pieces of classical music. For just $1.99 a piece you can finally buy top-of-the-line sheet music on-line at Puresheetmusic.com and choose from over 10.000 classics covering from Bach and Beethoven to piano and guitar.
»Puresheetmusic.com gives you more for less. The sheets are of the highest quality, the price is a bargain and once you pay it's yours for the rest of your life,« says Lars Christian Lundholm, founder of Puresheetmusic.com and a professional musician and composer.

»The idea was generated from my own personal frustration. I had a hard time finding sheet music that lived up to my high standards and when I finally did, it was always overpriced. After constantly hearing the same frustrations from my colleagues worldwide I decided to do something about it and created this new concept,« says Lars Christian Lundholm.

The future iTunes of sheet music - just click, pay and play
Puresheetmusic.com has set out to become the iTunes of sheet music as new sheets are constantly

www.Plus500.dk/Handelsplatforme

Kvinder Søger Mænd
Registrer dig gratis! Den nye dating site for Danmark.
www.prime-date.dk

Norske fjorde
Oplev det bedste av fjordene. Besøg Norges rejseguide!
VisitNorway.com/Fjordene

” is register while the “Page Focus” is on Up button the application will select the previous article(when you first arrive the page The selected Topic will be Arts and the first article will be the selected one in consequence you can’t go up till you go down).

7.6.2. Appendix G

Training sequence diagram