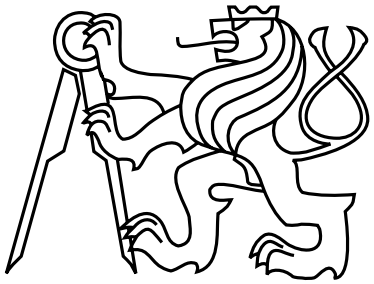
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Possibilities of Text Input for Handicapped People

Klára Fiedlerová

Supervisor: Ing. Petr Novák, Ph.D.

Study Programme: Electrical Engineering and Informatics

Field of Study: Computer Science — Software Engineering

May 10, 2012

Aknowledgements

I would like to thank Petr Novák, the supervisor of my thesis, that he always made time for me when I needed it. Big thanks also go to the employees and clients of the Jedlička Institute for young disabled people in Prague, for kindly allowing me to test my application with them, and for providing useful feedback.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Doksy on May 10, 2012 .....

Abstract

This thesis deals with the problem of designing a text entry system for the EasyControl application, which is being developed by the NIT Group at the Faculty of Electrical Engineering of Czech Technical University in Prague. EasyControl is targeted to disabled people and consists of a set of applications (such as an e-mail client, a text editor, or simple games) and special hardware input devices to control these applications. The text entry system developed in this thesis replaces a very simple software keyboard that has been used in EasyControl so far and is designed to be used with the special hardware devices.

Five of nine designed "keyboard" types were implemented and are part of the developed software module for EasyControl. The resulting text entry system is highly configurable. The layouts of the keyboards (the number of keys, their arrangement, and the content and output of each individual key) as well as the visual appearance of the application can be fully customized to the user's needs. Apart from the description of the designed keyboards, the developed application, and its testing, this thesis also contains a survey of some existing text entry systems.

Abstrakt

Tato diplomová práce řeší problém návrhu systému pro vstup textu pro aplikaci jménem EasyControl, vyvíjenou na Fakultě elektrotechnické Českého vysokého učení technického v Praze skupinou NIT. Aplikace EasyControl je určena pro hendikepované uživatele a skládá se ze sady aplikací (jako je e-mailový klient, textový editor, nebo jednoduché hry) a speciálních hardwarových vstupních zařízení, sloužících k ovládání těchto aplikací. Systém pro vstup textu vyvinutý v rámci této práce nahrazuje zatím používanou softwarovou klávesnici a je navržený především pro použití se zmíněnými speciálními hardwarovými zařízeními.

Pět z devíti navržených typů "klávesnic" bylo implementováno a je tedy součástí výsledného softwarového modulu pro EasyControl. Navržená aplikace je vysoce konfigurovatelná. Jak rozložení klávesnice (počet a rozmístění kláves, obsah a výstup jednotlivých kláves), tak vzhled aplikace lze plně přizpůsobit uživateli. Tato práce kromě popisu navržených typů klávesnic, implementované aplikace a jejího testování obsahuje také přehled již existujících aplikací použitelných pro vstup textu do počítače.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Every computer user usually has two or three favorite devices to control his or her computer. Has it ever happened to you that you for some reason could not use one of these devices? Perhaps your mouse or touchpad stopped working, or you spilled a coffee on your keyboard. Everyone who has this experience knows how hard it can be to replace the device we are used to with an alternative way to control our computer. All of sudden you realize how dependent you are on that device, and how inconvenient it is to use something else — for example a software keyboard, about whose existence you had not even known before that coffee accident. Luckily, for most of us this is just a temporary situation. Once you get your mouse fixed, or replace the damaged keyboard for a new one, you can get back to your usual way of controlling the computer.

However, there are people who are permanently (or at least for very long) in such situation. They have disabilities that prevent them from being able to use "mainstream" computer peripherals. These disabilities include limited hand movement, fine motor skills disorders, limited reaction ability etc. The default configuration of a typical computer and its operating system usually provides insufficient support for such users. If you have ever tried o use a mouse-controlled software keyboard, designed to look like the standard physical keyboard, you know that it is not easy to write some text longer than a few words with it. Handicapped people are faced with such inconveniences every day.

Good news for these people is that nowadays there are quite a lot of applications out there to help them access the computer. There are alternative hardware devices (such as switches, joysticks, and head pointers) that provide alternative ways to control a computer, and software applications that allow the user to input text or control the pointer using these devices. The application developed in this thesis is one of them.

The goal of this thesis is to design several types of text entry applications, that will be referred to as "keyboards" for simplicity. Some of them should then be implemented and tested. The implemented keyboards will be part of a software module that will be used in an already existing application called EasyControl, developed by NIT (Nature Inspired Technologies) research group at Czech Technical University in Prague. [11, 18]

EasyControl is a universal control system targeted to users with limited movement and/or reaction abilities. The system is designed to be able communicate with alternative hardware devices, and provides a group of applications for the users, from a set of computer games

to an application that controls a wheelchair. Some of the applications that are part of EasyControl require a text input from the user (e.g., the Mail application). A very basic software keyboard is currently used, which is however not adapted to the differences between the various hardware devices that can be connected to the system. The aim of this thesis is to replace the existing keyboard with a new application that would better suit the needs of each individual user and the device he or she prefers to use.

1.1 Organization of This Thesis

In chapter 2, the analysis of the context for the developed application is presented. The EasyControl system and its special hardware input devices are introduced. The objectives of this thesis are presented in more detail, and some related work is mentioned.

Chapter 3 contains a survey of already existing text entry systems. Both traditional on-screen keyboards and applications providing alternative ways of entering text are included.

The designed keyboard types are presented in chapter 4. This chapter also contains summary of guidelines for designing user interface, created based on published research articles. Special attention is paid to targeting the design to disabled users.

Implementation of the application is discussed in chapter 5. The application's architecture and the integration in the existing EasyControl system is described. How the implemented keyboards can be controlled and configured is explained. The chapter also contains discussion of possible future work. The chapter does not focus on the low-level details of the implementation, since the full documentation of the source code is available as a standalone attachment of this thesis.

Chapter 6 contains the evaluation of the implemented application. Advantages and disadvantages of each keyboard types are discussed in this chapter. Observations from the usability testing in the Jedlička Institute for physically disabled young people in Prague are also presented.

Finally, the conclusions of this work are presented in chapter 7.

Appendix A contains the user manual for the application. Content of the CD attached to this document can be found in appendix B.

Chapter 2

Background

This chapter presents analysis of the background for the text entry system that is developed in this thesis. The system is intended to be integrated in the EasyControl application. This chapter provides a brief description of this application, as well as of the input devices EasyControl works with. It also discusses target users, and summarizes the objectives of this thesis.

2.1 Target Users

The most important thing when developing a software application, or when doing any work on Human-Computer Interaction (HCI) field in general, is to know who are the users and what are their abilities. The application can thus be aware of their needs, and be less influenced by the developer's projections. This section contains an overview of the users of the EasyControl system whose needs will have to be taken into account in the design and implementation phases of this thesis. The characteristics of the target users comes mainly from the cooperation of the EasyControl developers (one of whose is the supervisor of this thesis) with the Jedlička Institute for physically disabled young people in Prague.

EasyControl system [11, 18] was designed for people whose movement and/or reaction abilities are limited. That means they are not able to achieve the same speed and accuracy as healthy people. The disabilities include cerebral palsy (a disorder that affects muscle tone, movement, and motor skills), muscular dystrophy (progressive weakening of muscles), fine motor skills disorders, poor vision, nervous system disorders, lack of concentration, injuries, and many others. These users therefore usually cannot use ordinary computer devices such as keyboard or mouse. However, their abilities can vary widely. For example, some persons are able to move their hands, but not to type. Other persons cannot move hand at all, but they are able to press something. Last but not least, there are also persons who are not able to move anything else than their eyes.

The state of the disabled people is either permanent or temporary. Temporarily disabled people could be for example those recovering after a backbone surgery. They need to communicate quickly (for example, with their doctor); they do not want to learn a lot of features of the application – the simplicity of the communication takes priority over the adaptation of the application to the user. Unlike the temporarily disabled people, the people that are

disabled permanently are usually not expected to make any considerable progress in what they are physically able to do. On the other hand, permanently disabled people are willing to learn how to work with applications designed to help them. They do not mind if mastering the application takes some time, because it is worth for them to take the time to learn it. However, that does not mean it is less important for the application to be adjustable for the concrete person's needs.

There are a lot of negative factors that prevent disabled users to access computers in the way other people access them. One of the biggest problems is using standard input devices. Persons with cerebral palsy, for example, are in general unable to use their fingers and can perform only small hand movements. That means they cannot precisely point and click on a button on a screen. If they are provided with a clicking device, they can perform presses, but the time between the press and release may become very long, because the action is complicated for them. [17] Such input may result in a drag-and-drop action (instead of a simple "click") in interfaces that are not prepared this. It must be taken into account that disabled users can make a lot of mistakes, and it may not always be practical to require correcting all of them.

Most severely handicapped persons have also visual problems [17]. Such impairments are greatly varied. Some disabled persons may have problems to distinguish foreground from background due to insufficient colour contrast, while another persons' eyes may be unable to simultaneously process areas with large differences in brightness. Things such as the size of the font, colours, or number of controls (buttons) usually become very important for users with poor vision.

Speed should play no role in any application disabled people use. There should be no time constraints, or at least the application should be configurable to allow adjusting all time limits to the user's abilities. That is necessary, because each disabled person has different needs. The time a disabled person needs to perform an action may even vary for him- or herself, depending for example on his or her current health status, or mood. The application should therefore try to reduce the number of steps (e.g., number of key presses) and the time to achieve a goal.

It is important to mention that the target users of the text entry application developed in this thesis are people with physical disabilities, but not mental disorders. A completely different approach would probably be required if the application would be targeted to mentally handicapped users. The EasyControl application was not designed for such users, and therefore neither the text entry system will focus on these users.

2.2 Using Alternative Input Devices

Disabled people typically need alternative devices to be able to control their computer. Special devices are available for these users, designed for the different abilities the users have. The list of all types of alternative computer peripherals would of course be very long. The project called *AbilityHub* [12] provides an interesting overview of such devices. However, the alternative input devices that can be used with the EasyControl application will be more important for this thesis, because those will have to be taken into account in the implementation.

Even a regular PC mouse can serve as an alternative device to a physical keyboard. It is necessary that the user is able to press the mouse buttons, or is at least able to move the mouse. That is unfortunately not always possible for the users. A joystick, belonging to a group of devices detecting a hand or a head movement, can be used instead. It can be controlled for example with the user's palm. The joystick still requires a good precision of the hand movement, because the user must have the ability to select all the directions. The selected direction can be either a signal for the pointer to move, or it can be mapped to some other action. If the user cannot move his or her hands, but is able to somehow perform a press and a release, he or she can use special switches (a set of switches or a single switch). It is then up to the software application to interpret the switch presses as actions. There are also devices based on eye-tracking or eye-blinking, that are usually used by the disabled people who cannot use any other devices.

In general, the input devices that can be used to control a computer can be divided into two groups. The first group, *continuous input devices*, are devices that can move a pointer on the screen continuously. A mouse, a trackball, or gaze-controlled devices belong to this group. These devices also typically need to be able to perform a "select action", so they usually contain a button or two for that. The second group, *discrete input devices*, are devices that move the "cursor" using discrete actions, usually presses of buttons or switches, but actions like changing the deviation of the device from its initial position can also serve this purpose. Select action is performed similarly, i.e. there is usually a dedicated button or direction of the deviation for that. Moving the cursor with these devices makes it "jump" from one item to another, which means that some item is always selected. This is not true for the continuous input devices, which may be seen as their disadvantage. The continuous input devices are usually faster, but they require better coordination between the eyes and the body of the user (e.g., a hand) than the discrete input devices.

The EasyControl system was designed to work especially with the alternative input devices developed as a part of the research project by the NIT Group [18]. This means that also the text entry (keyboard) module that will be part of the whole application should focus on these devices. Currently, there are six devices available:

- *Finger Switches*, a set of "buttons" that can be controlled by pressing them. How the switches are pressed does not matter, which means the user can use fingers as well as toes, palm, head, chin, etc.
- The joystick — a lever-controlled device. It is capable of detecting up to eight directions of the lever deviation.
- A device detecting a movement/inclination of the part of a body it is attached to. The behavior is similar to the joystick — the device detects the deviation from the horizontal position, i.e. the direction of the inclination (up, down, left, right, etc.).
- A device detecting a pressure (applied by a palm, fingers, foot, or toes). The behavior is similar to the switches.
- The "universal glove" — a glove with an inclination detector and detectors of finger movements. Changing the inclination can for example be used to control the mouse pointer, while bending a finger can trigger some specified action.
- IR controller containing various buttons.

The EasyControl system includes hardware interface for connecting these input devices

to a computer via USB (Universal Serial Bus), and a software tool for communicating with the hardware and configuring it. Using this tool, it is possible to configure the devices and their output.

2.3 EasyControl Application

Let us know look at the other part of the EasyControl system, which is a set of applications intended for the target users. [23] For this thesis, the current implementation of its built-in software keyboard is especially relevant.

Applications available for the current user are listed in, and can be run from, the "Tool-Bar", a software component that serves a starting point for the user (it is similar to "quick launch" controls known from operating system's desktops). This component also allows the user to switch between the keyboard simulation and the mouse.

Currently, the available applications include a text editor, a mail client, "OnLine" chat (a Jabber client), an application to control a cell phone via BlueTooth interface, and various logic games (for example Sudoku, Lights Out, Pairs, Sokoban, and others). An application that controls home environment of the user (opening/closing doors and windows, controlling the TV and radio, etc.) is another, standalone, part of the EasyControl system, as well as the application that controls the user's wheelchair.

Obviously, only some of these applications require a text input. If they do, there is a simple keyboard available. Figure 2.1 shows how the keyboard looks like in the mail application.

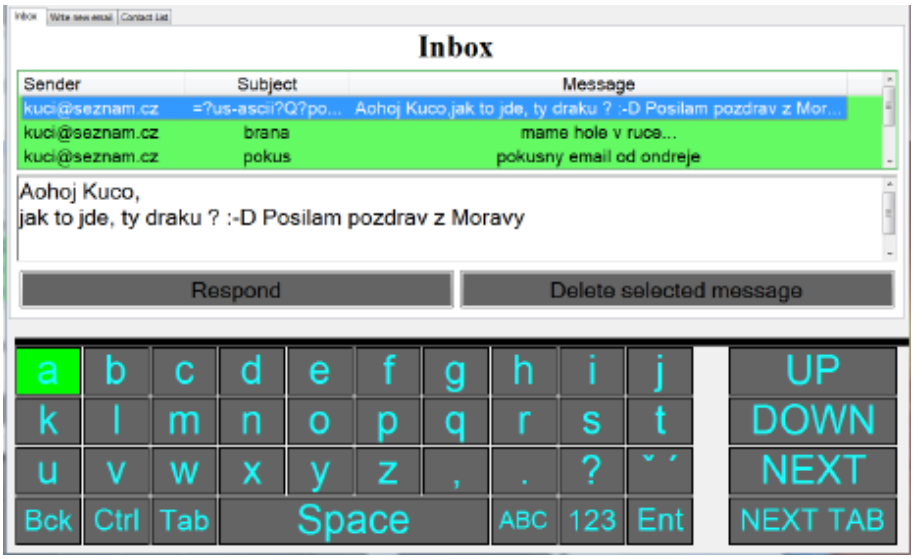


Figure 2.1: EasyControl: The mail application

Besides the keys that are used to write text, the keyboard also contains keys for special actions (see figure 2.1, the special keys are placed on the right) that allow the user to control the user interface of the application. Each application can use a different set of actions. In

the mail application shown in the figure, there are actions to move up and down in the list of received messages, to move the focus on the next focusable element, and to switch between the tabs (Inbox, New Mail, Contact List). That means the whole application is controlled by the software keyboard. The performance of the user (speed, accuracy) therefore depends on how easy it is for him or her to control the keyboard.

There is currently only one type of keyboard available — a standard keyboard that has its keys arranged in a rectangular grid, similar to a physical keyboard. The layout of the keyboard uses alphabetically ordered letters, instead of the traditional QWERTY, as the authors believed this layout will be more intuitive for novice users who are not used to writing with a traditional keyboard.

The keyboard has very limited configuration options. Basically, only some colours can be changed (colour of the font, background colour of the keys, and the background of the whole keyboard). However, it is good that the keyboard allows changing at least these properties. The ability to change the colours is important, for example to achieve higher contrast for users with poor vision. It can also simply be more pleasant for the users to use a keyboard that has their favourite colours.

2.4 Problem Description

Providing only one keyboard type is very insufficient, considering the number of different input devices that can be used together with the EasyControl application. If alternative hardware devices to control the application are offered, it would be appropriate to also offer alternative types of "software devices", i.e. a text entry application suitable for each device. These alternative text entry applications, that we can call "keyboards" for simplicity, could be then designed to better suit the concrete input device and its interface — i.e., the way how the device is controlled by the user, what outputs it can generate, the number of buttons it has, etc.

A good thing for the software keyboard developed in this thesis is that the hardware devices are supposed to filter most of the input noise. Such noise can be caused for example by the user accidentally hitting a key multiple times, as a result of his hands shaking. That means the keyboard application can assume the input events coming from the device are fine, and does not have to do any filtering by itself.

2.5 Objectives of This Work

As we have seen, the keyboard is a part of a large system that contains various applications that serve very diverse purposes. The keyboard can, however, be needed by any of them. That means it has to have an interface that is universal, not dependent on the context in which the keyboard is used.

The objective of this thesis is to improve the built-in keyboard application for the EasyControl system. The new "text entry module" should be as universal as possible, and also highly configurable. This configuration should not only include settings of the keyboard's appearance (colours, size of keys, etc.), but also the layout (arrangement) of the keys. What keys will be part of the keyboard should be as much as possible left to the user's preferences.

Each user should be provided with an option to use the type keyboard that suits him the best, which means the integration of the keyboard settings with user profiles is crucial.

In the future, word prediction and abbreviation expansion features are planned to be integrated in the keyboard. The implementation should therefore be prepared for this, and allow easy addition of these extensions.

Several keyboard types should be designed, so that there is at least one for each of the alternative hardware input devices that can be currently used together with EasyControl (see section 2.2). The implementation of the text entry application therefore has to be focused on these special devices.

2.6 Related Work

Some work has already been done to investigate the word prediction feature that could be included in the text entry module of the EasyControl application. The investigation, including an implementation of a prototype system that tests the word prediction system, was done as a part of Martin Vogal's Master's Thesis [32] (in Czech). The prediction system described in the thesis includes not only word prediction, but also a prediction of whole sentences. It contains some interesting ideas. However, the developed software system itself has very limited possibilities of integration, as it does not provide a clean interface to communicate with. This is the reason why this prediction software, as it is now, cannot be yet integrated with the keyboard — neither the one already existing, nor the one that is developed in this thesis.

A lot of standalone software keyboards (or other text entry applications) exist. As the survey of similar applications is very important for this thesis, an entire chapter was dedicated to it. General explanation of the principle, as well as some examples of software keyboards, both commercial and free, is provided in chapter 3.

Chapter 3

Virtual Keyboards

Applications used for text entry can be divided to two groups. The first group includes applications based on the traditional keyboard, i.e. the keys are arranged in a grid and typing is done by selecting them individually, one after another. The other group consists of applications that take advantage of some other ways of typing. The alternative typing possibilities vary widely, and include for example using keys that contain multiple letters, typing with gestures, or text entry based solely on word prediction.

With the expansion of "smart" devices with touch screens, software applications for text entry without the need for a hardware keyboard had to be developed. Variety of alternatives to the traditional keyboard appeared. Every touch-screen device typically needs to contain some application that allows text entry. The keyboards with traditional grid layout are usually still the default option, but the user has the possibility to use one of the various applications allowing to input text differently than by traditional "typing". A lot of the alternative text entry applications are based on gestures, trying to be more closer to handwriting than to typing on a keyboard.

A disadvantage of most standalone software keyboards is that they run in a "non-focusable" window, and send the output to the window that currently has the focus. That allows them to be used with almost any application, but at the same time it causes problems to some users, especially those that are not able to control the computer freely, or use some alternative input device that has a limited capabilities in terms of controlling the computer. For example, it can happen that a pop-up message appears in the desktop tray. As a result, the window that the user has currently worked with loses the focus, which consequently means the user cannot any longer send the keyboard output to it. But what if the user is unable to switch the focus back to the text editor window? In such situation, an assistance of another person is usually necessary, which is of course a big drawback.

In the next two sections, some representatives of both groups of text entry applications will be introduced. These sections are of course by no means meant to be comprehensive, but will rather contain applications that utilize some interesting ideas. All pictures are used with permission from the authors.

3.1 On-Screen Keyboards

As mentioned earlier, this group of applications display what could be called a "traditional keyboard", i.e. a software keyboard based on the physical keyboard device that is used to send a text input to a computer. Because the keyboard is displayed to the user on a computer screen, the common term "on-screen keyboard" (OSK) is used for such type of application.

Most operating systems contain a simple on-screen keyboard that can be used if the user for some reason cannot use the hardware keyboard. Typing with such keyboards is usually very slow, and they provide only basic functionalities. They can serve as a quick temporary solution when the hardware keyboard is not available. There are of course also some standalone applications that provide a simple software keyboard. Every touch-screen device usually contains such keyboard as well.

The next level of applications are commercial, freeware, or open-source applications that display an on-screen keyboard as well, but contain additional features such as text prediction, configurable keyboard layouts, possibility to turn on the "dwell clicking" and so on. There is a better chance the typing will be more convenient and faster with such keyboards.

Eight examples of on-screen keyboard applications follow, with emphasis on those that contain text prediction or other interesting features.

3.1.1 KeyStrokes 4

KeyStrokes® 4 is a commercial application developed by Origin Instruments [22], which is available for MAC OS. The application can be controlled by a mouse, trackball, head pointer, or other mouse emulators. Figure 3.1 shows the keyboard in action.

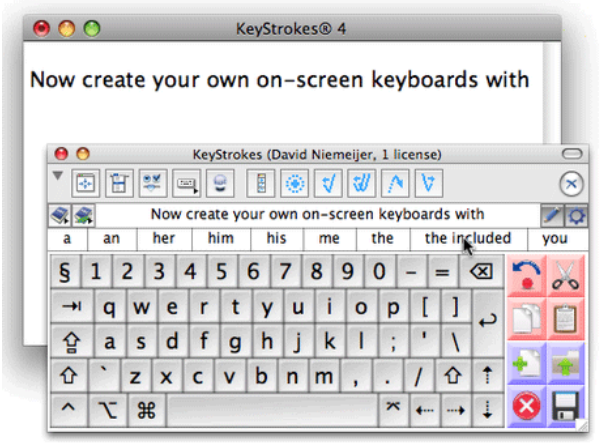


Figure 3.1: KeyStrokes® 4 — the application window (used with permission from AssistiveWare)

Keystrokes 4 is highly configurable and contains many features that make typing easier. The application offers multiple keyboard layouts to choose from, and also allows the user to design his own layouts using a designer called LayoutKitchen™. A dwell clicking feature

is also included. It has a programmable period of time and clear indication of the current mouse button state. The click action and default click action can be changed (to single click, double click, right click, drag, etc.) by dwell-clicking on one of the special keyboard controls. The keyboard has an integrated word completion system. The system includes next word prediction, multi-word prediction, learning with automatic spell-checking, and a dictionary editor. The predicted word(s) can be written by clicking on the appropriate button from row of words that is displayed directly above the keyboard. Another feature to mention is an audio feedback (including speaking the typed words). The application is localized in several world languages.

The KeyStrokes 4 keyboard is meant to be used not only for text entry, but to control the computer in general — it is claimed to work with any standard Macintosh application.

The main advantages of the KeyStrokes 4 are that it contains a lot of features, the application is highly configurable, and the interface provides a great visual feedback to the user. However, the amount of features is reflected in the price of the application, which is $299 for one license. That is a downside of this software, since the cost can be too high for some users. Another disadvantage is that it is available only for Mac operating systems.

3.1.2 Grid Keys

Grid Keys is a virtual keyboard for Windows OS. It allows the user to use standard Windows applications without the need to use hardware keyboard or even a mouse. The application can be controlled by alternative pointing devices (such as a trackball, joystick, head pointer, and touch screen) or switches. [25]

Figure 3.2: Grid Keys — the application window [25]

The Grid Keys application window is shown in figure 3.2. A word completion is integrated in the keyboard (by means of special buttons containing the words). The dwell clicking can be turned on, so the application can be controlled by a user who is unable to perform clicks. The keyboard screens (called grids) are configurable, meaning not only that the placement of the keys can be changed, but special keys can be added or removed as necessary to suit the control device and the current situation in which the keyboard is used (writing, browsing web, mouse pointer control, etc.). The application also offers a speech output and is available in several languages, including Czech. It can be purchased for $350.

The price of the Grid Keys application is very high, which makes it unavailable for some users. Recently, just before this thesis was completed, the Grid Keys software was discontinued, and it is now only possible to get a complete computer-access software package from Sensory Software, called The Grid 2 [26], that is similar to the EasyControl application for which the text entry system implemented in this thesis is developed, and that also contains a highly configurable keyboard.

3.1.3 OnScreen with WordComplete

OnScreen is a collection of on-screen keyboard utilities for individuals who can use any pointing device or switch, and need an on-screen keyboard as their primary text input device. [5]

The keyboard window is shown in figure 3.3. The keys are arranged in a traditional grid, with the possibility to change the layout to another predefined one. The keys and key functions can be remapped and the application also provides a way to define keystroke macros. Word completion is included — a special column of buttons with five suggested words can be found on the left-hand side of the keyboard. The words are not *predicted*; the application builds a dictionary only from words typed by the user. However, the word completion system contains helpful features such as auto-spacing, auto-capitalization, suffixes, undo option, and few others.

Figure 3.3: OnScreen with WordComplete — the keyboard window [5]

The application can be controlled by a single switch, which can be of great benefit for disabled users. The feature supporting a single switch is called CrossScanner. It presents a vertical scan (scanning down the screen), so the user can select the Y coordinate for the action (which can be a single click, double click, or drag). The horizontal scan then begins and continues until the next switch hit, which selects the X coordinate and triggers the action.

OnScreen with WordComplete works under Windows operating systems and costs $199. The official web page, cited earlier, claims the application works with any software that runs under Windows. A disadvantage of the keyboard is that the visual appearance of its user interface cannot be changed. The default layout looks very old-fashioned, and some users might have problems with its low contrast (black text on grey background). There is a possibility to change the appearance of the keyboard's interface with an additional software that the user would have to buy, which would increase the cost of the application.

3.1.4 WiViK

WiViK® is a virtual keyboard software developed by Holland Bloorview Kids Rehabilitation Hospital. [14] It provides access to MS Windows applications using any pointing device (mouse, trackball, touchpad, and others). The application offers a default, traditional-looking layout (see figure 3.4), which can be customized by the user (the keys can be rearranged, keys with new functionality can be created, colours and sizes of keys can be changed). However, the layouts are defined in a text file, and there is no graphical interface for editing them. The user therefore has to be skilled enough to be able to customize the layouts.

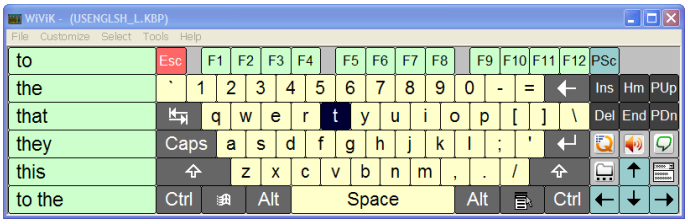


Figure 3.4: WiViK® — the application window [14]

The world completion system consists of many features, including word prediction and abbreviation expansion (two or three letters followed by the Spacebar or Enter key are expanded into phrases of full sentences). The word prediction is adaptive based on how the user combines words, and the application also contains topic-specific vocabularies.

Other features of WiViK® include dwell clicking (highlight can move or scan across the keyboard) with customizable scanning strategy, speech output, and localization in four languages. Single-user license costs $350.

3.1.5 Click-N-Type

Click-N-Type is a free, full-featured on-screen keyboard for MS Windows operating systems. It was developed directly with emphasis on disabled users and is claimed to work with all Windows applications that can run in a window. [16]

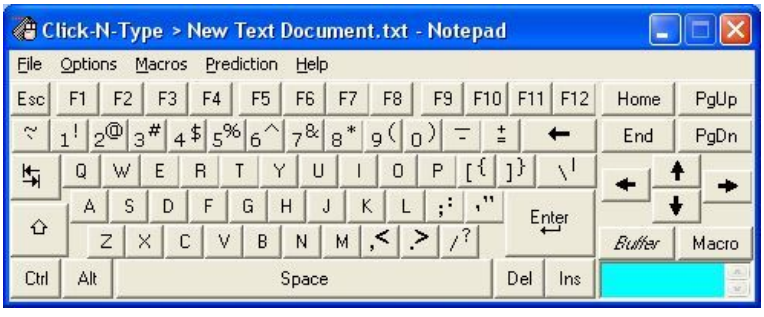


Figure 3.5: Click-N-Type — the application window [16]

Figure 3.5 shows the keyboard window. The application offers multiple keyboard layouts to choose from. A keyboard designer is available, allowing the user to create custom layouts. Support for macros is provided as well.

Optionally, a word prediction window can be opened and attached to the keyboard, offering list of predicted words. The words can be typed by clicking them.

Auto clicking with adjustable timer is also available. The application contains a scanning feature, which is three-phase: first, a rectangle moves across the keyboard, allowing user to select the area containing desired key. In the next two phases, vertical and horizontal lines move inside the selected area. The three-phase approach is meant to speed up the scanning process. The application has many other features (for example a speech output), the list of which can be found on its website [16]. There are about 40 language packs available for the Click-N-Type keyboard. A big advantage of the Click-N-Type keyboard is that it is free, and still it contains features of commercial keyboard applications.

3.1.6 The Fitaly One-Finger Keyboard

The Fitaly keyboard application is designed for Pocket PCs and tablet devices to minimize the pen or finger travel. It supports MS Windows operating systems, both the standard and mobile flavours. [4] The application website contains the reasoning for the keyboard layout, which is claimed by the authors to be more ergonomic than standard keyboard layouts such as QWERTY, because the keys are ordered in a way that the most frequently used letters (in English language) are placed in the middle of the keyboard (see figure 3.6).

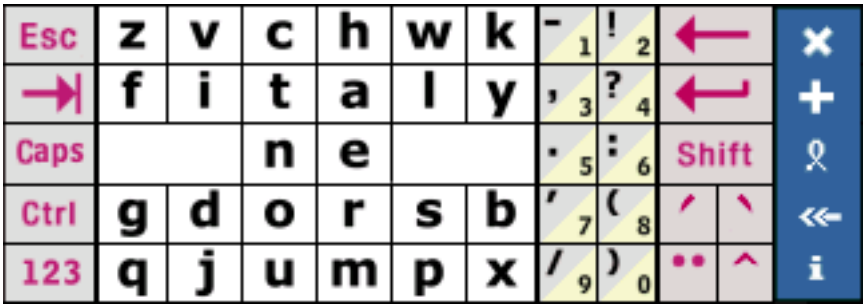


Figure 3.6: Fitaly One-Finger Keyboard [4]

The Fitaly keyboard also supports word completion, abbreviation expansion, and macros. It has an interesting feature called “sliding”. Sliding happens when the user taps on a key and moves the pen (or finger) sufficiently far before releasing it – the direction then determines the available actions (called slides) for the given key. These actions are then displayed in a list above the keyboard. An action can be anything from writing another letter to launching an application. If there is only one action available, it is performed immediately – this way the sliding can be used to write for example capital letters or accents.

The Fitaly keyboard for Pocket PC costs $29, while the version for a Tablet PC costs $49. The keyboard has a full Unicode support and contains dictionaries for multiple world languages. A disadvantage, from a disabled user’s point of view, is that the keyboard is not

very customizable (e.g. the colours of keys cannot be changed), and that there is no support for automatic scanning or dwell clicking.

3.1.7 SwiftKey X

SwiftKey X is an application for the Android OS, meaning it is designed to be used on small devices with touch screens. [7] The keyboard layout therefore has to be minimalistic to fit the limited available space (see figure 3.7).

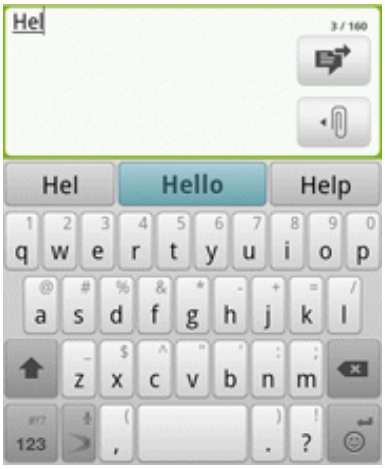


Figure 3.7: SwiftKey X application [7]

Word prediction is used to speed up the text entry. The prediction system uses the context of the sentence to predict three words that could be used next. The most probable of them is displayed in the middle in the row of predicted words. The user does not even have to type any letter to get the predicted words.

The user can customize the appearance of the application by using themes for the keyboard.

There is a similar application, SwiftKey Tablet X, intended to be used with tablets. The SwiftKey X is available in 35 world languages. It is designed for smart phones and tablets, so it cannot be used on an ordinary PC. It also does not contain any features for disabled users.

3.1.8 ThickButtons

The last example of an "on-screen" keyboard in this section is an application called ThickButtons, available for devices using Android OS. [1]

The application is targeted to touch screens. Its keyboard is a simple grid, but the size of its keys changes dynamically based on next-letter prediction. See figure 3.8 that shows the prediction in action.

While the user is typing, the prediction is used to enlarge and highlight the useful buttons and shrink the other ones. That means the prediction is an integral part of the keyboard, not

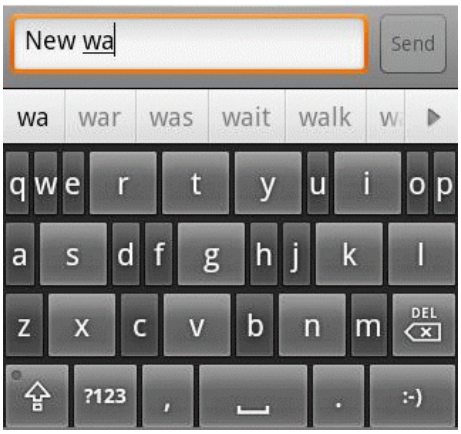


Figure 3.8: ThickButtons application [1]

just "one of the features". The application also contains a row of predicted words that can be selected directly. The enlarging and shrinking keys is an interesting feature that could be utilized by applications for disabled users.

3.2 Alternative Typing

Even though the on-screen keyboard applications differ in the appearance and their features, they are based on the same typing principle. Keys are arranged on the screen (usually in a grid) and each key has to be selected (typed) individually. Optionally, word completion/prediction can be used to increase the speed of typing. However, there are alternatives to this traditional approach, some of which will be introduced in this sections. These alternatives reflect the needs of special groups of users (touch-screen users, people with disabilities, etc.), or were simply developed to offer a different (faster, more convenient) way how to type without a physical keyboard. One of the disadvantages of many (but not all) of the alternative keyboards is that the user may have to learn a special "alphabet", because the letters are arranged in a different way than on a traditional QWERTY keyboard. Sometimes even a completely new way of writing has to be learned.

3.2.1 DKey

DKey is a keyboard similar to the one that can be found on mobile phones. The keyboard consists of a grid of twelve keys, nine of them containing three or four letters. The application is designed for people with disabilities (it contains fewer keys than traditional keyboard, allowing the users to use special input devices like number pads and keypads), but can of course be used by anyone. [21] Besides the devices already mentioned, the keyboard can be controlled with a mouse, or switch devices via other software.

The writing itself works like the T9 predictive writing, available on most mobile phones: the user presses keys with the letters he wants to use, the software looks up which words can

be constructed from those keys, and displays them in a list. The space key is used to finish the word.

Dwell clicking (switch scanning) and text-to-speech features are provided. Colours of the keyboard can also be customized. DKey runs under MS Windows and is a free open source software.

3.2.2 TapTap Keyboard

TapTap keyboard is in fact very similar to the on-screen keyboards that were presented in section 3.1. The only difference is that this keyboard tries to minimize the space it takes by using less keys. Reason for this is that the application is targeted to mobile devices that have small screens. In order to be able to write all letters, each key has to contain more than one of them. A letter can be typed with two taps — first tap selects a group of letters, and the second finally selects the desired letter that is then written. See figure 3.9 that shows an example of both possible "levels" of the keyboard.

Figure 3.9: Example of two possible states of the TapTap keyboard [31]

The author admits the typing with this application is not very fast, but on the other the application has the advantage of taking little space. [31] Because there are fewer keys, they can be bigger than those on traditional software keyboards for mobile devices).

This application unfortunately provides no word prediction or support for disabled users (such as dwell clicking). However, it is free (licensed under GNU GPL).

3.2.3 MessagEase Onscreen Keyboard

MessagEase Onscreen keyboard was also developed mainly for devices with a touch screen, but it can run on a PC as well. The keyboard is optimized to reduce the stylus or finger movement, which eases text entry and increases the speed of writing. [19] It supports MS Windows OS, and can be used with touch screen or mouse. Figure 3.10 shows the keyboard layout.

Most frequent letters (those that are yellow) can be typed directly with a single tap. Less frequent letters (those that are white) are typed by dragging the pointer from the square which contains the letter to a neighbouring square in an appropriate direction. For example, letter "B" is written by dragging from "O" to "R", letter "X" is written by dragging from "I" to "O", etc.

The application offers no word completion or customization of the keyboard's appearance, except that it is possible to make it transparent. It is localized in multiple languages, and the keyboard is customizable for basically any alphabet. The price of the application depends on the target device, but does not exceed $25.

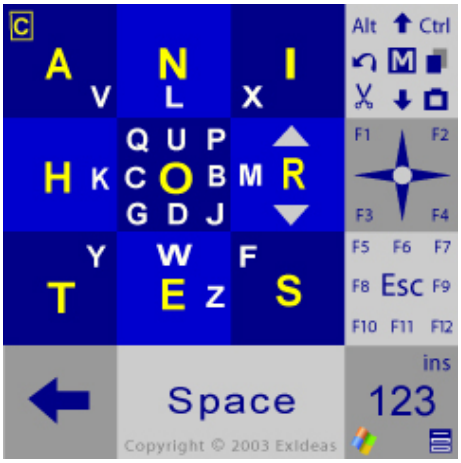


Figure 3.10: MessagEase Onscreen Keyboard [10]

3.2.4 Clicker

Clicker is a commercial application intended for children to help them write without a keyboard. [8] It contains many possibilities of text input, from constructing sentences from predefined words to writing using pictures. Figure 3.11 shows an example of the application window.

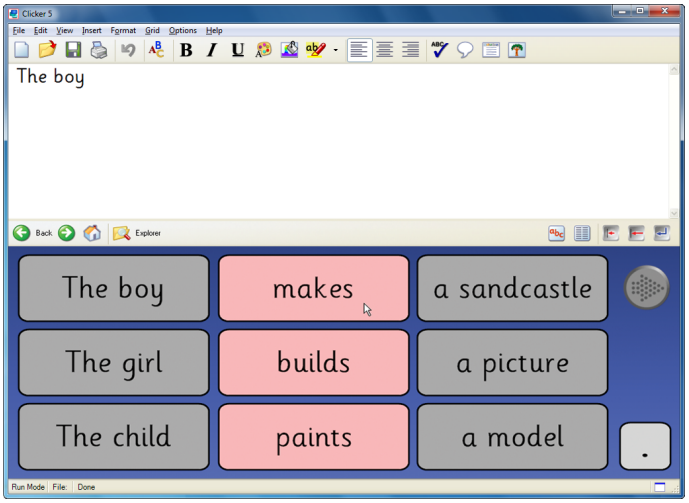


Figure 3.11: Writing sentences with Clicker (reproduced by kind permission of Crick Software `http://www.cricksoft.com`)

The so-called "cells", that are used to form the keyboard, are customizable, i.e. they can be set to display any content. For example, the application can be configured to display words that are most frequently needed by the user. Not only children, but also people with disabilities could utilize such functionality. A traditional keyboard layout (with each cell

containing a letter, digit or symbol) is also available.

Clicker has an integrated support for dwell clicking and can be used with mouse and other pointing devices, or switches. Single-user license for Clicker 6 costs $150.

3.2.5 Quikwriting

Quikwriting is an innovative writing application designed to be used on pen-based devices, and is available for free. The writing is performed without a need to tap or even pick the stylus up off the screen surface, and it is based on gestures.

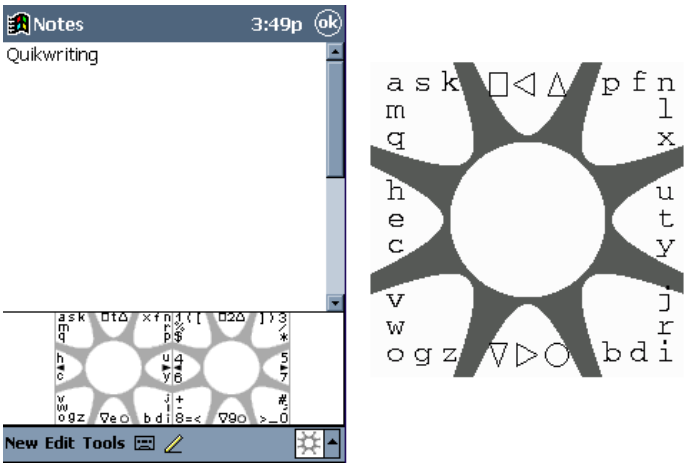


Figure 3.12: Quikwriting — the application window and a detail of the keyboard pattern [15]

A Technote published at the ACM UIST'98 conference summarizes the way how Quikwriting works [24]. See figure 3.12: each character is positioned in one of the eight outer zones (its major zone) and also at some relative position within this zone (its minor zone). This position corresponds to how that character is drawn. To write a character, the stylus is moved from resting zone (the centre) to the character's major zone, then to the character's minor zone, and finally back to resting zone. For example, the letter "f" is drawn by moving from the central zone to the north-west zone, then left to the north zone, and then back to the centre. Emulators are available on the project's website [15]. The Quikwriting application has been used by MERU (Medical Engineering Resource Unit) in the UK to provide children with special needs with a way how to easier access their computers. [15]

3.2.6 8pen

8pen, an application for Android OS, provides a way how to write that is similar to handwriting. No tapping (clicking) is needed to write a character. A screenshot of the application is shown in figure 3.13. The writing is done by drawing "loops", which is a natural way how people move with a finger or pen, if they are asked to draw a continuous line, according to the research carried out by the authors [3].

Figure 3.13: 8pen application [3]

See figure 3.14 for an explanation how the writing works. The canvas of the "keyboard" is divided by four edges into four sectors. The letters lined up along one side of the edge indicate an assignment of these letters to the loops starting in the sector to that side of the edge, and leaving the sector on the side of their alignment. Each loop crosses the number of edges determined by the position of the letter in the row. The loop for the innermost (first) letter crosses one edge (its own) before returning to the centre, the loop for the outermost (fourth) letter has to cross all four edge lines. The figure shows examples of three different loops (for letters "a", "d", and "f"). A space is automatically inserted when the user lifts the finger from the centre region.

The application supports all devices capable of detecting gestures (mobile phones with touch screens, digital cameras, game controllers, ...). It also offers features such as word suggestions and auto-complete. Multiple language layouts are available. The application can be purchased for $0.99.

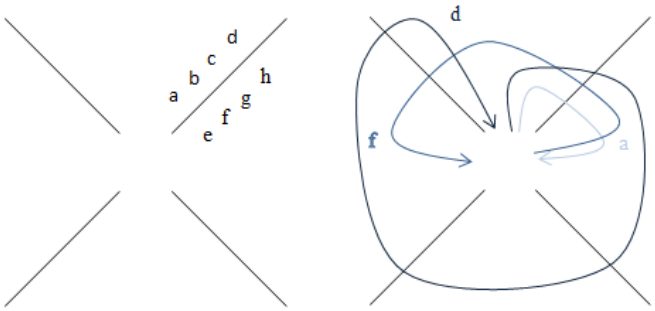


Figure 3.14: How writing with 8pen works [3]

3.2.7 Dasher

Dasher is a text entry application based on word prediction. It allows for fast writing just by moving the pointer, and is therefore ideal to be used for example with head pointers and

similar alternative devices. It is free (licensed under GNU GPL), and supports operating systems including MS Windows, Mac OS X, and Android. Figure 3.15 shows the application window.

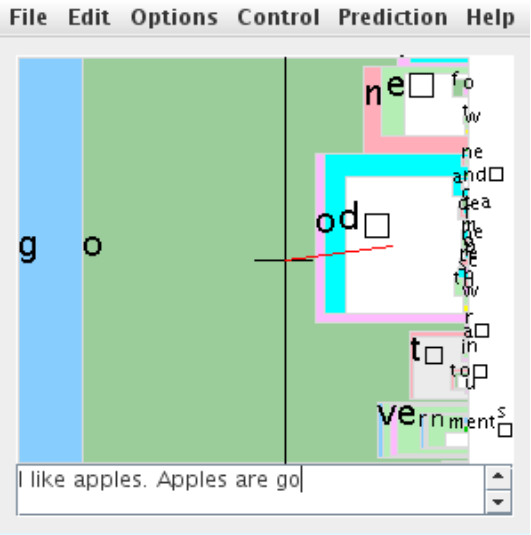


Figure 3.15: Dasher application [9]

The letters are arranged in the alphabetical order from top to bottom in a column on the right. Any point on the screen corresponds to a letter, the "path" to the point corresponds to a piece of text. The user points where he wants to go, and the display zooms in there, i.e. the user chooses what he writes by choosing where to zoom. The more the user zooms in, the longer piece of text has been written. Probable pieces of text (determined by word prediction) are given more space. Moreover, colours are used to better distinguish the character areas. The project's web page [9] contains a demo application and more detailed explanation of how to work with the application.

3.2.8 EdgeWrite

EdgeWrite is a stroke-based (gesture-based) text entry system developed by Jacob Wobbrock [34]. The application has very simple design, which consists of a square input area with clearly marked corners. The letters are written by "drawing" in this area. The order in which the four corners of the area are entered during the writing determines the character being made. There is a special alphabet that defines how each character has to be written. Even though the strokes look similar to the real letters they correspond to, they are not exactly the same, and therefore the alphabet needs to be learned by the user (see figure 3.16).

The EdgeWrite application is designed to be used with a wide range of devices from joysticks and touchpads, to minimal, four-key devices. EdgeWrite contains an interesting word-completion system. If the user does a little loop at the end of the stroke, the system displays four predicted words along the edges of the input area that begin with the letter

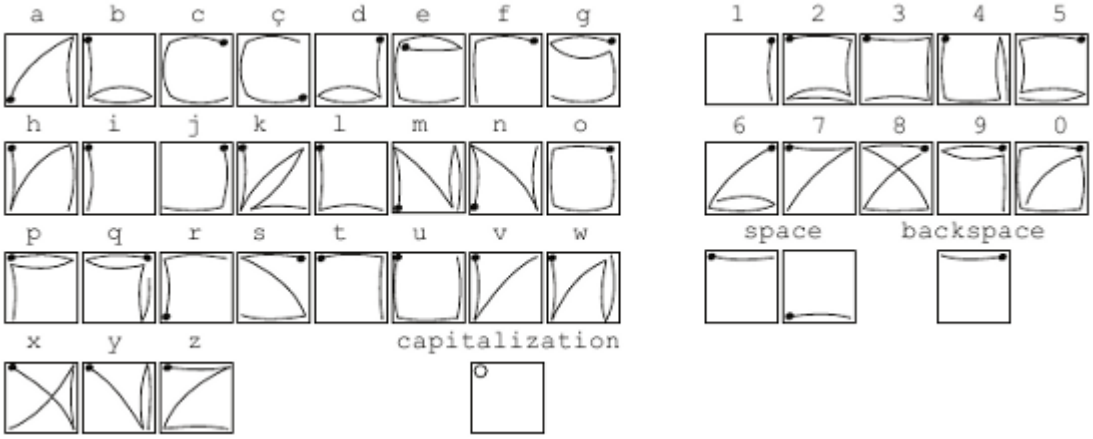


Figure 3.16: Part of the EdgeWrite alphabet [2] (the heavy dot marks the beginning of the stroke)

that is being entered. The user can then draw a line towards the edge on which the desired word is displayed, which results in the word to be written. The same set of letters is always displayed for each stroke, which allows the user to memorize the gestures for common words.

The EdgeWrite application can be downloaded for free from the project's website [2], and is also available for developers as a DLL (Dynamic-Link Library).

Chapter 4

Design

In this chapter, several keyboard designs will be introduced. The keyboards are different not only in how they look like, but, more importantly, in the way how they are controlled. That means each keyboard type will be suitable for a slightly different group of users and different input devices that EasyControl offers. This was an intention when designing the keyboards, as the variety of choices makes it easier for the user to find a keyboard he or she will be best comfortable with. Before introducing each the designed keyboards, some important consideration and usability rules will be mentioned, because those also influenced the process of designing the keyboards.

4.1 Requirements for the User Interface

The goal of the design process was to come up with a set of keyboards, among which every user of the EasyControl application could choose one that is suitable for him or her. When designing any user interface in general, there are a lot of factors that need to be taken into account. Considering the that the keyboard being developed in this thesis is targeted to disable users, it is clear that there will also be some special requirements (for discussion on the target users, see chapter 2).

4.1.1 Guidelines for Designing User Interface

Jakob Nielsen's Ten Usability Heuristics [20] is a well-known set of principles that help the designers and developers make better user interfaces. The list of these heuristics is presented below, together with examples of how the rules relate to the particular case of a software keyboard user interface:

1. **Visibility of system status.** The users should be always informed about what is going on. For example, if a key has been pressed, some feedback should be given immediately. In some situations, like when the user is writing a letter, it can be sufficient that the letter immediately appears in the output window. However, stronger feedback is often needed. For example, the background of the key could be changed for a short period time right after the key has been pressed. In case there are keys

that can change the "mode" of the keyboard, there should be an indication of which mode is currently active (for example, a *CapsLock* would activate the mode of writing capital letters).

2. **Match between system and the real world.** The keyboard should not use any terms or symbols that are unknown for the user. Let us give some examples. If the user is not familiar with common symbols for special keyboard keys (like ↩ for the Enter key, ⇑ for the Shift key, etc.), they should not be used and be rather replaced with something else that will communicate what the key does (e.g. with a symbol that will be clearer for the user, an icon, a group of letters, or even a whole word, if necessary). The usage of terms from the computer technology area falls into the same category. Another example can be users that have never used a physical keyboard. Such users will be confused with the QWERTY layout, whereas the alphabetical order of the letters can be more natural for them. On the other hand, if the user is familiar with physical keyboards, the opposite will be true. Last but not least, the software should speak the same language as the users. That means not only that all texts should be localized, but also that the keyboard should provide a way how to write all special symbols used in the user's written language.

3. **User control and freedom.** It should be always clear for the user how to "undo" an action, or how to return to the application's default state. For example, if the user accidentally switches the keyboard mode (e.g., from lower-case to upper-case letters), there should be a clearly marked key that will allow him to get back to the previous state without having to write any letter in the current mode.

4. **Consistency and standards.** The application should be consistent in all respects. For example, if the keyboard offers more than one layout (e.g., one for letters and one for symbols), and some of the keys are the same for both layouts (e.g., the "Delete" key), then the shared keys should be placed on the same position in both layouts. If the platform standard is to have a button that closes the application in the upper-right corner, and the keyboard also offers a similar action, the key with this action should also be in the upper-right corner of the keyboard's interface. If there are custom actions on the keyboard (e.g., a "Send" key), their names should be clear and use standard terms, so that the user does not have to wonder what the action does.

5. **Error prevention.** The interface should be designed in a way that it is clear to the user what to do to achieve his or her goal, and that it does not often happen that the user is confused and makes a mistake. Since the user interface of a keyboard is not very complex, it is unlikely that there will be a need for displaying error messages to the user. When it comes to the typing itself, features like spell checking and word prediction can help to reduce errors (such as typos and misspelled words).

6. **Recognition rather than recall.** All important actions that are relevant in the current context should be visible to the user. The application should not require the user to remember anything, but rather it should contain a mechanism how to present that information to the user. For example, if the user presses the "CapsLock" key on the keyboard to write an upper-case letter, an indication that this key has been pressed

should appear and be present until the mode is deactivated. If the indication was not there, the user would have to remember that he has pressed the key.

7. **Flexibility and efficiency of use.** It should be able to adjust the keyboard application for each individual user. It is also seen as a benefit if there are two or more ways how to do the same thing — some may be more suitable for a novice user, some for an expert. The user can then choose himself which way of manipulating the application he prefers. If there are any timeouts used by the keyboard (e.g., automatic confirmation of a key without the need of clicking it), they have to be adjustable, because every user has different reaction abilities. In general, the more configurable the interface is, the better.

8. **Aesthetic and minimalist design.** The application's interface should only contain what is really needed for the user to see. The design of the interface should be simple rather than cluttered, and there should not be any unnecessary "eye-catching" effects (for example, if there is something that moves in the interface, it constantly attracts user's eyes — such animations should therefore be used only if there is a good reason for it). Colours used in the application should go well together. A keyboard application is really a "functional" piece of software, so it is very important that it does not distract the user from his work. However, at the same time, the interface should be pleasant to look at.

9. **Help users recognize, diagnose, and recover from errors.** If there is a need to display an error message, the description of the error should be clear and should not use the application's "internal language", such as error codes. The user should be informed about what happened and what the application is going to do now, and possibly also advised what to do next to recover from the error.

10. **Help and documentation.** Even though it would be best if the application's interface was completely intuitive for the user, sometimes this cannot be achieved (e.g., if the user has never worked with such type of application, or if the application provides novel concepts of controlling the interface). In any case, documentation and a user manual should always be available to the user. The manual should be written in a "how-to" style, so that the user can easily repeat the described steps. Also, no documentation should be too large.

Another useful list of principles for interface design, called *The First Principles of Interaction Design*, was published by Bruce Tognazzini [30]. Some interesting pieces of advice from the list include:

- **Just colour is not enough.** If a colour is used in the interface to indicate something important for the user (i.e., it does not just have a decoration function), there should always be another (secondary) cue that indicates the same thing.

- **Avoid uniformity. Make objects that act differently look different.** If all buttons in the application look exactly the same, the user will expect them to have the same behaviour. For example, if all buttons that form a keyboard in the keyboard application write a letter, except for one that switches the keyboard layout from letters

to symbols, this one button should be clearly distinguishable from the others. At the same time, the interface must be consistent, i.e., in our example, the user still has to be able to tell what is and what is not a button.

- **Fitts' Law**. The Fitts' Law states, that *the time to acquire a target is a function of the distance to and size of the target.* In case the application is controlled with a pointing device, it is important that all the controls in the application are easy to reach. That means the buttons should be big enough, and the most important ones should be placed on the edges of the interface.

Last set of principles that will be examined in this section are *Eight Golden Rules of Interface Design* [28] by another well-known expert on HCI, Ben Shneiderman. These rules often overlap with Nielsen's heuristics, proving their importance and universality. Again, these principles will be applied to the concrete case of designing a keyboard application.

1. **Strive for consistency.** Actions should have identical names across the whole application. For example, if there is an action called "Clear" in the text entry application that causes all text to be deleted, this action should not be called, say, "Delete All" at some other place in the application. Similarly, both "Close" and "Exit" names should not be used, if they actually represent the same action — only one of them has to be chosen. This also applies to terminology used in user messages, to interface layout, meaning of colours, or capitalization. This is said to be the most violated principle.

2. **Enable frequent users to use shortcuts.** It is helpful to provide special commands for expert users, in order to enable increasing the pace of interaction. These "shortcut" actions can be assigned to special keys, making them optional to use. In the case of the software keyboard, for example, we could provide an option for the expert users to add more keys on the keyboard. If the user needs to write some symbol often, he could be allowed to add it to the standard layout. The keyboard would then probably lose its simplicity, needed for novice users, but it would help the experienced user to speed up the writing.

3. **Offer informative feedback.** There should be some system feedback for every action. How substantial the feedback will be should depend on the importance and frequency of use of the corresponding action. See the first rule of Nielsen's heuristics presented earlier in this section for examples.

4. **Design dialogue to yield closure.** If a sequence of actions is required, they should be organized in groups, and there should be a clear feedback each time a group is completed. This feedback is a signal for the user that he does not have to "worry" about the completed actions any more, and that he can concentrate on the next step. This would for example apply to all configuration dialogues of the text entry application.

5. **Offer simple error handling.** The system should be designed so that it was not possible for the user to make a serious error in the first place. If it happens that an error is made, the system should be able to detect it, and offer simple mechanisms for handling the error. For example, if the keyboard application supported user profiles that are saved in a file, and the system would not be able to load the file, the system

should offer the user an option to use a default profile, or to choose from other existing profiles. Such error could also be handled by the system straight away, without even asking the user for what to do (this could be a better approach for novice users, or users who are not aware that such functionality is there, because somebody else configured the system for them).

6. **Permit easy reversal of actions.** If the user can rely on the fact that each step he takes can be undone, it will be much less stressful for him or her to work with the application. It can also encourage him or her to explore the interface. This principle is contained in Nielsen's third rule (see above).

7. **Support internal locus of control.** The user, and especially an experienced one, feels more comfortable if he has a sense of having full control over the system, rather than being just a responder to system's actions. The interface should therefore give the user sufficient freedom. However, the designer also has to be aware that too much freedom can have a negative effect (see the Autonomy principle in Tognazzini's list [30]).

8. **Reduce short-term memory load.** It is important that the interface does not "overload" the user with information or any other signals (e.g., motions) the user has to process. At same time, however, things that belong together should be displayed in the same window, so that the user did not have to remember them while he navigates through the interface. Example of application of this principle in the software keyboard can be the appearance of its keys. If there are not just keys for writing letters, but also keys for other actions on the keyboard, those keys should not be overused. Only a few of these special keys for the most frequent actions should be shown, and the rest could be somehow hidden, and become visible only upon user's request.

The user interface design and usability guidelines mentioned in this section contain a lot of practical advice and apply to any user interface in general. However, if the application is targeted to users that have some kind of disability, some of the design guidelines may become even more crucial to implement, and also new issues with the interface design can arise. The next section summarizes some important findings mentioned in several articles from authors that conducted research in this area, focusing on those that will be especially important for designing the keyboard application.

4.1.2 Usability and Disabled Users

There are differences in abilities even among "normal" (not disabled) users. In fact, there is nothing like a "normal" user, because every person is unique. However, the degree of variability in "normal" users' abilities is still rather small than in case of the disabled users. There are a lot of kinds of impairments that a user can have (as already discussed in chapter 2), and it can therefore be very hard to provide "general guidelines" that would apply to all of these users. That, however, does not mean we should give up on research in this area, it only means that usual approaches to designing user interface may not be sufficient if the target users are handicapped.

There are several approaches to providing people with special needs access to computer technologies. A recent article by Jacob Wobbrock et al. [33] provides a brief summary of

these approaches. They range from *assistive technologies*, that are mainly concerned with how to "fit" disabled users to standard technologies by means of add-ons inserted between the user and the system, to *universal usability*.

Universal usability was described by Ben Shneiderman as the aim for making the information and communication services available to everyone, regardless of their knowledge, education, physical or cognitive abilities, experience, and so forth [27]. The three main challenges, as seen by Shneiderman, are:

- **Technology variety.** A broad range of hardware, software, and network access needs to be supported.
- **User diversity.** The interfaces have to accommodate users with different skills, knowledge, age, gender, disabilities, literacy, culture, etc.
- **Gaps in user knowledge.** There must be a way to bridge the gap between what users know and what they need to know.

The universal design approach therefore does not make any special provisions for the disabled users, it just sees them as one part of the wide group of all users. Shneiderman believes that such approach can benefit all users — they will get interfaces that are much more flexible than those designed with traditional approaches. In fact, even a "normal" user can get into a situation where he will be faced with some disabling conditions (such conditions include having to type in gloves, working in a sunlight or noisy environment, or dealing with a broken input device). User interface that is adaptive can then help all users equally well.

The article by Jacob Wobbrock [33] promotes an approached called *ability-based design*, which takes the idea of universal design further. The idea of ability-based design is to orient to "what a person *can* do", rather than to "what a person *cannot* do" or "what *everyone* can do". The question then is how a system can be made to fit the abilities of the user, no matter who that user is. In the ideal case, the system should be so flexible that it would allow people to use it without requiring them to alter their bodies, knowledge, or behaviour. In reality, it will not probably be possible for the user interface designer to create a system that would meet 100 percent of this vision. However, it is possible to achieve it at least to some extent. What is more, designing the system with the focus on abilities in mind changes the designer's thinking and helps him create a product that will be more flexible. The main principle to follow to achieve the goals of ability-based design is to make the system *highly adaptive*. The interfaces may be either self-adaptive (often in response to performance or context) or user-adaptable. The first case involves continuous monitoring, modelling, and/or predicting the user's performance, and can therefore lead to quite complex implementations. It is also important that all these adaptations remain transparent to the user — the system has to allow him to view, change, or even discard the adaptations. Even though the ability-based design principles encourage the designers to use standard hardware components for the system, they do acknowledge that some users, especially those with severe disabilities, may have to use custom components. The biggest challenge of developing an ability-based interface is modelling the user's abilities, because the variability among users with impairments is very high, and therefore conventional user models do not suffice in this case.

The article called Designing User Interfaces for Severely Handicapped Persons by J. B. Lopes [17] emphasizes the importance of configurability of the interface as well, and also mentions the need to include tools to manage user profiles. All the other features of the interface have to be built on these essential requirements. The areas, to which the designer should pay special attention, include, but are not limited to:

- size, colour, and number of items shown on the screen
- minimal user interface design (for example, animations are not recommended for most users)
- different levels of difficulty, and alternative ways to perform user tasks
- support for wide range of input devices
- various output models (graphics, sound, speech)
- strong feedback mechanism

Research results from the area of Human-Computer Interaction from several authors were presented in this section. They provide a lot of interesting findings that could (and will) be utilized when designing the text entry application in this thesis. It seems that the most important principle to follow, promoted by all the HCI experts, is to make the interface as adaptable to the user's preferences as possible, especially when it is targeted to disabled persons. In other words, the interface has to be aware of the wide range of personal characteristics of its users.

4.2 Applying the Requirements

Let us now look at how all the guidelines presented in the previous section can be utilized in the design process of the text entry system (called "keyboard" for simplicity) developed in this thesis. Even though some examples of applying the guidelines to a software keyboard interface were already given, those examples were rather general ideas not yet applied to a real interface. This section will explain the decisions and compromises that were made, and on which the various keyboard designs are built.

It is clear from the guidelines presented in the previous section that the keyboards have to be highly configurable. What can be configurable and what cannot will of course also depend on how the keyboard is controlled and how does it look like. The minimum available options should include:

- **Size of the keys.** It should be possible to enlarge or shrink the size of keys.
- **Colours.** All background colours, colour of the font, or any other colours used on the keyboard should be configurable to allow achieving any level of contrast and/or brightness.
- **Timeouts.** If timeouts are used (i.e., some event occurs after a period of time elapses), they have to be adjustable.
- **Number of keys.** The keyboard should offer an option to remove or add extra keys, and thus make either simplified or more complex layouts.
- **Order of keys.** The keyboard should either offer various layouts (an alphabetical layout, the QWERTY layout, etc.), or it should be possible to rearrange the keys on the keyboard to make the keyboard suit the user's needs.

- **Support for icons.** If a lot of text is used, the interface may become hard to orient in. The use of icons can help in such cases, as they are also able to communicate the meaning of actions. The keyboard should therefore support keys with icons.
- **User profiles.** The keyboard has to be able to manage user profiles, in order to allow different users share the application on the same computer and not having to reconfigure it from scratch every time they run it.

As you can see, a great part of the requirements is connected to the layout of the keyboard, i.e. what keys will be present on the keyboard, and what will be displayed on those keys. This leads us to the idea of allowing the user to configure the whole layout. This will make the application extremely flexible. The user will be able to decide exactly which keys he wants to have on the keyboard, what will be displayed on those keys (a letter, a symbol, a word, an icon, ...), and what their output will be. Besides the keys that actually "write" something, the application should of course also provide special action keys (such as Delete, Shift, etc.). However, the application should let the user decide if he wants to use these keys, and where he wants to put them in the keyboard layout. Therefore, it will be completely up to the user (or, better said, the person who will be creating the keyboard layout) how the keyboard layout will look like.

An important thing to note here is that the configuration of the EasyControl application, and thus also of the keyboard, will be typically done by another person, not by the disabled users themselves. This is necessary, because it would not be wise to require the disabled users to configure for example the input devices (e.g., mapping the input signals to actions in the application), as those settings are quite complicated.

The configuration of the keyboard application can be simply added to already existing configuration mechanism. The application will thus be user-adaptable, not self-adaptive. It would of course be possible to make the keyboard self-adaptive in some ways, but as an assistance of another person is needed for the configuration anyway, the "user-adaptable" solution can be chosen as less demanding on the implementation.

Besides the configurability, the other important requirement is that the keyboard types have to be designed with respect to the input devices that can be used together with the EasyControl application (for the list of these devices, see section 2.2). The designed keyboard types should vary, so that there is at least one keyboard suitable for each input device. The following characteristics will therefore be important:

- the minimum number of "buttons" (actions) that will be needed to control the keyboard (for example, navigation to four directions requires four different buttons, confirmation of a key requires one button, etc.)
- whether or not it will be required that the user moves the pointer
- whether or not it will be required that the users performs clicks/presses

4.3 Designed Keyboard Types

This section contains description of proposed keyboard types. There is an explanation of how each "keyboard" works, how it is supposed to be controlled, and which devices are

recommended to be used with it (with respect to the alternative input devices for the Easy-Control application). The terms "continuous" and "discrete" input devices will be used in the following sections (see section 2.2 for an explanation of the difference). When explaining what the input device needs to be able to control the keyboard, the term "button" will be used for simplicity, even though it is not technically correct — a device need not have any buttons, but still it can generate similar output as devices that have buttons. For example, moving a joystick to the right can have the same effect as a button press, and the application can therefore interpret it similarly.

Almost all of the keyboards (except those that can be controlled *solely* by continuous movement of the pointer) can be used together with some kind of auto-scan and auto-click functions. These functions can replace either the need to navigate through the layout or the need to press something to select a key. One possible implementation of auto-scanning is automatic jumping (i.e., moving the focus) from one item on the screen to another. If the user wants to select an item, he has to wait until the cursor gets there and then press a button. Auto-selection (confirmation) usually means that a key is selected if the user stays on it (i.e., does not move anywhere) for some specified period of time. A disadvantage of such functions is that they usually slow down the typing. However, if the user is not able to use more than one button, or cannot perform presses or clicks, it may be necessary to use these features.

There is a picture for each keyboard design. Note the pictures have the form of drafts, in order to illustrate the idea of the design, and not focus on details. They are not meant to represent a detailed specification for the implementation. That is why they do not provide any details such as colours, fonts, and shapes. The set of keys is also only illustrative. After all, these details should be configurable in the actual implementation.

It also should be noted that it is supposed that the keyboards support a "layout switching" functionality. That means that it is be possible to change the set of buttons on the keyboard, for example from the default set containing letters and digits, to another set containing symbols. This functionality allows to include all necessary letters, symbols, and actions in the keyboard without having to display them at once, and thus making too large and complex layouts. For some keyboard types, this functionality can even help reducing the number of buttons that are needed to control the keyboard.

In each of the following sections, there is also a discussion on how to incorporate prediction in the particular keyboard layout (note that only a next-word prediction will be taken into account, not for example a prediction of whole sentences). As mentioned earlier in this thesis, the keyboards must be prepared for the integration of a prediction system, which will be most likely implemented in the future (a prototype already exists, as mentioned in section 2.6). Prediction can help the disabled users not only to speed up their typing, but also prevent typing errors.

4.3.1 Grid Keyboard

Design of the *Grid Keyboard* is based on the appearance of a traditional keyboard. The keys are arranged in a grid. Number of columns and rows is supposed to be configurable, which makes this layout very flexible. For simplicity, all keys have the same size. However,

it would also be acceptable to make some keys bigger (i.e. longer and/or wider, so that they span multiple cells of the grid). Figure 4.1 shows the keyboard layout.

Figure 4.1: Grid keyboard

How It Works

Controlling the *Grid Keyboard* should be intuitive for everyone who has ever used a physical or virtual keyboard: a key is highlighted when the user navigates to it, and then confirmation (e.g., a click or button press) is needed to select the key.

Suitable Devices

This keyboard can be controlled either with a continuous or discrete device. The number of buttons that are needed to control the keyboard with a discrete input device can vary. One button is always needed for confirmation (selecting a key), and the rest is used for navigation. Typically, four buttons will be used to navigate up, down, left, and right. Additionally, more navigation actions can be used to allow the user to navigate to the diagonal directions. Another possibility is to use only one (or two) buttons for navigation — in such mode, pressing a button would cause the cursor to jump to the next (or previous) key. That means that minimum of two and maximum of nine buttons can be utilized. A joystick could also be used with this keyboard. If all eight directions were used for navigation, however, the confirmation would probably have to be done automatically, or the joystick would have to have an extra button for that.

Pros and Cons

Advantages of the *Grid Keyboard* are that it has very flexible layout, and it is intuitive to use (if the user has ever worked with a computer and used a keyboard). It can also be used with various devices, regardless if they provide continuous or discrete input. A disadvantage is that the navigation on the keyboard may become quite slow if discrete input device is used (for example, consider the layout shown in figure 4.1 — it takes ten moves to get from letter “I” to letter “S”, that means at least twelve button presses are needed just to write the word “is”).

How To Incorporate Prediction

The *Grid Keyboard* offers several possibilities of how to incorporate prediction. One option would be to add special "keys" on the keyboard, each key containing one predicted word. The user would have the option to write the predicted words by selecting these keys. An advantage of this solution is that the number of predicted words offered to the user could be configurable, and the keys could be placed anywhere on the keyboard — either in a separate row or column, or anywhere else among the other keys. Figure 4.2 shows an example of the layout including predicted words.

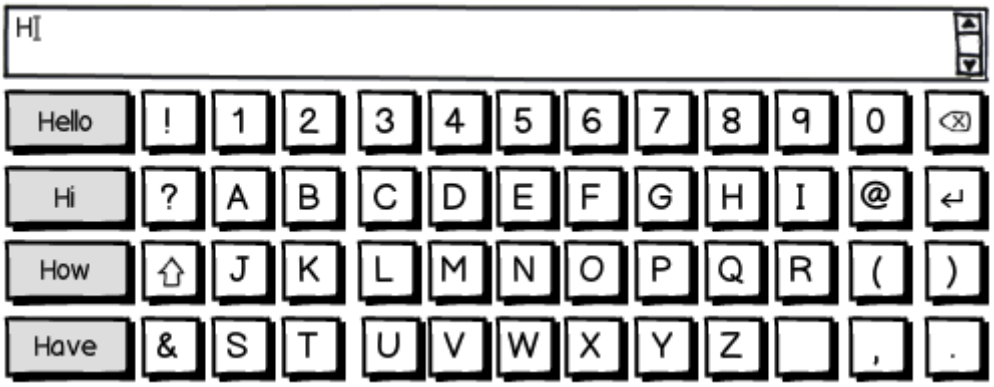


Figure 4.2: Grid keyboard with predicted words as special keys

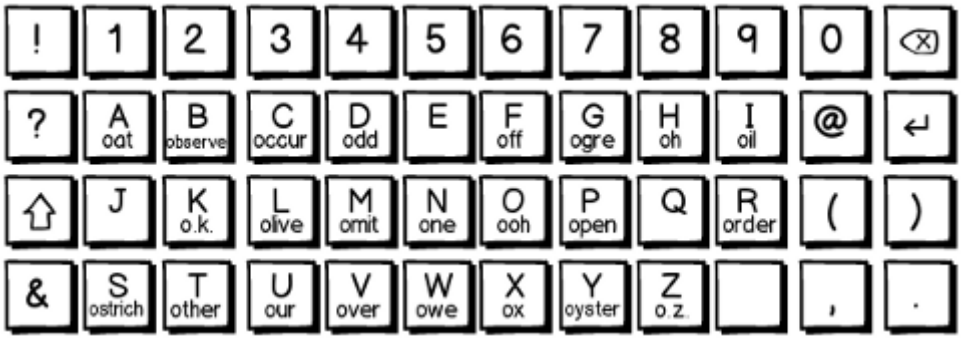


Figure 4.3: Grid keyboard with predicted words on keys with letters

Another option is to include the predicted words directly in the already existing keys, more precisely just on those with letters. A word would be placed on a key if the key contains the letter that would have to be selected next to write the predicted word. For example, if the user has already typed the letter "H", word "hello" would appear on key "E", word "house" would appear on key with "O", and so on. If the user then wanted to select the predicted word, he would have to select the desired key in some special way. For example, he could activate a special mode on the keyboard (something similar to the *CapsLock* mode), or he would have to hold the key for a longer time. Of course, it is possible that nothing can be

predicted for some letters. In that case, no word would be displayed on the keys with those letters. An example of how the keyboard could look like if the user began to type a word with the letter "O" is shown in figure 4.3. This way of incorporating the prediction has a disadvantage that it can happen that there will be a lot of predicted words for one letter, of which only one can be displayed on the corresponding key, but almost no words for the other letters. Therefore, the number of displayed words can be very limited. This problem could be solved by allowing to fill the "empty" keys with the rest of the predicted words, even though their letters do not match. However, the risk of this solution is that the words will be then too hard to find for the user.

4.3.2 Keyboard with Coordinates

The layout of the *Keyboard with Coordinates* (see figure 4.4) looks very similar to the *Grid Keyboard* described above. However, it significantly differs in the way how the keys are selected (see description below). The keys are arranged in a grid. Additionally, there is a number for each row and column. The upper left corner can be used to display status information.

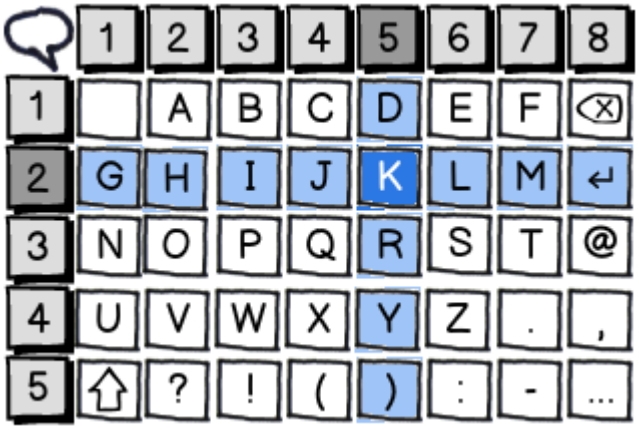


Figure 4.4: Keyboard with coordinates

How It Works

In a grid, each key is clearly defined by its position, i.e. its row and column number. This keyboard takes an advantage of this fact. Instead of having to navigate from one key to another, the user simply selects the *coordinates* of the key. Therefore, as less as two button presses are needed to select any key. In the example layout shown in the figure 4.4, the user has to press 2 (the row number) and then 5 (the column number) to write the letter "K". The key is selected automatically after the second coordinate is entered, so there is no need for any additional confirmation.

There is also another option how to control the keyboard. The rows and columns need not be selected directly by entering the coordinates, but they can as well be selected using a cursor

jumping from one row (column) to another. First, the keyboard allows navigation in rows (currently selected row can be for example highlighted with a different colour, or have thicker border), and the user navigates to the desired row using a *Next* action. Then he confirms the row selection, and the keyboard switches to navigation in columns. The user then selects a column and by confirming the selection, the key that is on the intersection of the selected row and column is "pressed". The number of button presses needed to select a key varies — the minimum is two (if the user wants to select the key on position $[1, 1]$), the maximum is $(numberOfRows - 1) + 1 + (numberOfColumns - 1) + 1 = numberOfColumns + numberOfRows$.

Suitable Devices

This keyboard was especially designed for the "Finger Switches", a five-button input device (see section 2.2 for description). Each switch is assigned a number from 1 to 5. That would, however, only allow using 5x5 grid. Such grid would not even be able to contain all letters. Additional numbers are therefore needed. This problem could of course be solved by adding more switches. Another option is to interpret long press or simultaneous press of two numbers as another number. In the latter case, number 6 could be achieved for example by pressing switches number 1 and 2 simultaneously, number 7 by pressing 1 and 3, and number 8 by pressing 1 and 4. The button number 1 would therefore serve as a special ("Shift") button, and would not output any number by itself (numbers 1–4 would be mapped to buttons 2–5). As a joystick also has eight output actions, it is suitable for this keyboard as well. If we wanted to control this keyboard with a device that has less buttons, we could simply remove some columns (or rows) from the layout.

However, some users may not be able to use switches in the described way, because it would be to difficult for them. Those users can use the other way of controlling the keyboard, i.e. selecting the row and column successively, utilizing a "confirm" action. This approach requires only two buttons — one for navigation, the other for confirmation.

If the user is able to control three buttons, we could use a little improvement in the navigation. The rows and columns could be selected "simultaneously" — one button would be used for selecting a rows, the second for selecting a column, and the third for confirmation. It would not matter in what order the row and the column are selected, the user could even for example navigate to a row (using the first button), then to a column (using the second button), and then again change the selection of the row (using the first button again), and only after that confirm the selection. Since it is not necessary to confirm the row selection, this way of controlling the keyboard would save the user one button press.

The *Keyboard with Coordinates* cannot be used with continuous input devices. If the user is able to use such device, a better choice for him or her is to use for example the *Grid Keyboard* (or any other keyboard type that can be controlled with continuous input device), because using this keyboard would not bring any benefit. The user would have to click on the row and column numbers, which would mean he has to do twice as much clicks as he would have to do with the *Grid Keyboard*.

Pros and Cons

An advantage of this keyboard is that it can provide a faster way how to type for users with discrete input devices, in comparison to the *Grid Keyboard*. With a five-button device, the user could use the traditional *Grid Keyboard*, but he would have to perform significantly more presses. Let us recall the example of writing the word "is". With the *Grid Keyboard* in figure 4.1, fourteen presses are needed to write this word (assuming the cursor had been placed on the space key before), while with the *Keyboard with Coordinates*, it takes only four if the direct selection of coordinates is used (in the situation in figure 4.4, the user would have to press 2 and 3 for "I", and 3 and 6 for "S"). If the successive selection of the row and column is used, the number of presses is higher — for the word "is", it would be twelve presses on a three-button device. If this "simple navigation" is used, the most frequently used keys should be placed near the upper-left corner of the keyboard, where the number of button presses to select a key is lowest.

One disadvantage of this keyboard is that the input device has to have sufficient number of keys if the direct selection of coordinates is to be used. Another disadvantage may be that the user has to memorize which button on the input device corresponds to which number in the application. This disadvantage could be mitigated by having the numbers written on the device, even though this may not be possible to do for some devices. The "simple navigation" has the disadvantage of being slower than the direct selection of coordinates, but on the other had, it is not that difficult and could cause less cognitive load for some users (since it provides clear visual feedback).

How To Incorporate Prediction

As the keyboard's grid of keys looks almost the same as the *Grid Keyboard*, the prediction could be incorporated in a very similar way (see section 4.3.1).

4.3.3 Shifting Keyboard

The *Shifting Keyboard* is very simple keyboard designed for users who may have problems with complex layouts or who can only use devices with limited number of buttons. The keyboard consists of a single row of keys, which means that it takes just a little space on the screen. The keyboard is shown in figure 4.5.

Figure 4.5: Shifting keyboard

How It Works

As its name suggests, the *Shifting Keyboard* is based on the idea of shifting a row of keys to find the desired key. A defined number of keys is visible on the screen and the rest is "hidden" on the left and/or right. The middle key is highlighted — this is the one that is selected if the user presses a select button on the input device. The user can move ("shift") the row of keys using corresponding navigation buttons. The simplest possibility is to shift the row by one key to the left or right. Optional navigation actions are to shift the row by the whole window (on the picture 4.5 it would be by nine keys), or move the focus to the very first or last key.

Suitable Devices

Basically any discrete input device can be used with this keyboard. One button is a minimum that the device has to provide (this button could be either used to confirm the highlighted key, while the navigation would be done automatically, or vice versa). At least two buttons are needed if no automatic scanning or confirmation is used. The keyboard can be controlled by up to seven buttons (one for confirmation, two for one-key shifting, two for whole window shifting, and two for jumping to the beginning or end).

Pros and Cons

Advantages of the *Shifting Keyboards* are that controlling it is very simple, and not many buttons are needed for it. It also saves a lot of space on the screen, which means the keys can be bigger than if any other keyboard was used. This can be a benefit for users with poor vision. The keyboard is also very suitable to be used together with the auto-scanning feature. If auto-scanning is used, some of the advantages of the various keyboard types are gone, because the automatic navigation is be done key-by-key, going in one direction. It therefore becomes unnecessary to have the whole big keyboard on the screen. Using the *Shifting Keyboard*, a lot of space can be saved. Also the cursor highlighting the keys does not "move" anywhere, it always stays in the middle, so the user does not have to follow it.

If the user controls the keyboard himself, the fact that only a part of the keys can be seen may become a disadvantage. The user has to know what direction to go to get to the desired key without seeing it well in advance. It is therefore crucial that the keys are ordered in a logical way for the user. This keyboard shares the disadvantage of "long path" between keys with the *Grid Keyboard* — the user typically has to press the navigation button(s) multiple times before he gets to the key he wants to select.

How To Incorporate Prediction

We could use the same idea as for the previous two keyboards, to include "special keys" for predicted words. However, this solution has a great disadvantage with this type of keyboard — the keys would be placed somewhere where the user could not possibly see them (for example, if the user types a letter that is in the middle of the keyboard, he typically cannot see keys that are placed at the beginning or at the end of the keyboard). If the predicted words could not be seen, the user would have to navigate to them every time, to see if there

is something he could use, which would not probably save him any key presses. Such solution is therefore not suitable for this type of keyboard.

A better option would be to display the predicted words always under the currently visible set of keys, in an extra row. The predicted word that could be then selected would be the one in the middle, under the currently highlighted key (see figure 4.6 for illustration). The question is what the user would need to do to write this word. We could add a special action for that, which would however require one more button on the input device. As this layout is meant to require as few buttons as possible, this option may not be acceptable in most situations. The predicted word could be rather written for example by holding the select button longer, or pressing it twice in a row during some defined time interval.

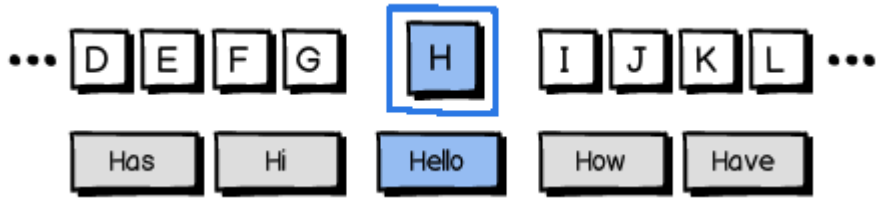


Figure 4.6: Shifting keyboard with prediction

4.3.4 3x3 Keyboard

The *3x3 Keyboard* has eight big tiles, arranged in a 3x3 grid, with an empty space in the middle. Each of these big tiles contains eight keys, arranged in a similar way. The keyboard layout is shown in figure 4.7.

How It Works

The *3x3 Keyboard* works on the principle of *selecting directions*. The position of each tile corresponds to one cardinal point (the upper-left tile is northwest, the upper tile is north, etc.). If a direction (cardinal point) is selected, the corresponding tile is "broken down" to its eight keys. These eight keys ("second level") replace the view of the whole keyboard. Figure 4.8 shows the keyboard after the west tile from figure 4.7 has been selected. In the second level, a concrete key can be selected. The keyboard then returns to the first level.

Note that the tiles/keys are selected by choosing the directions. There is no need to "navigate" to the desired tile (or key) and then press the select button. Only two direction selections are needed to select each key, similarly as in the case of the *Keyboard with Coordinates*.

Suitable Devices

This keyboard has to be used together with devices that are able to distinguish eight different directions. The joystick is a straightforward example of such device. However, also other devices, such as switches, can be used. Buttons just have to mapped to the eight cardinal

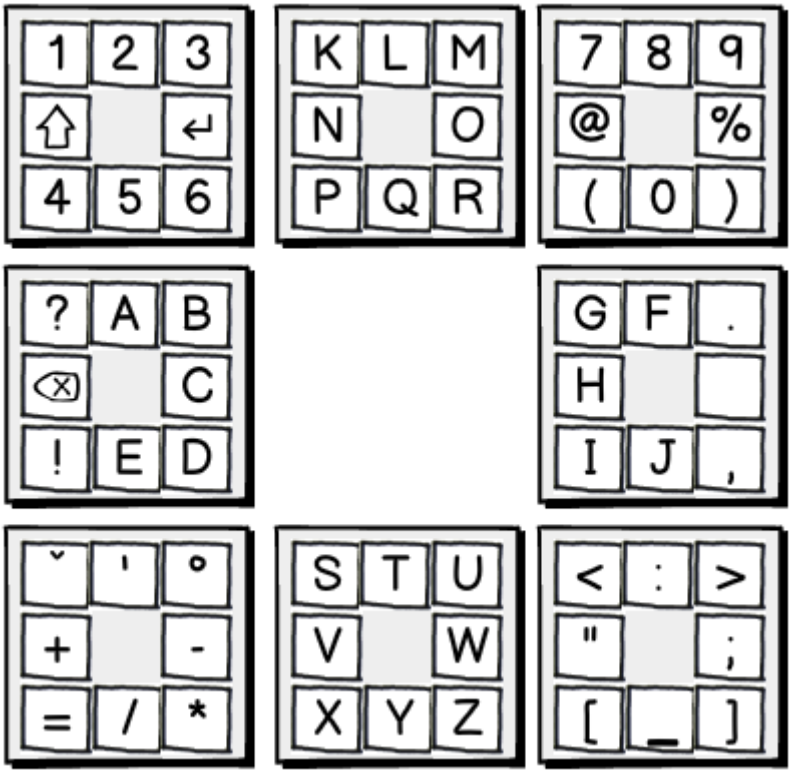


Figure 4.7: 3x3 keyboard — first level

points, and the user has to know mapping. It is possible to use continuous input devices with this keyboard as well. Selecting a direction can be performed by directly "clicking" the relevant tile or key.

Pros and Cons

The *3x3 Keyboard* has the advantage that it allows the user to select a key in just two button presses. The price for this is that more buttons are needed to control the keyboard. The keyboard is ideal to be used together with a joystick. An advantage of this keyboard layout also is that it can fit quite a lot of keys (64). That means that "switching the layout" will probably hardly be needed. Seeing everything at once eliminates the need to remember what keys are on the other layouts that can be switched to. On the other hand, the layout may be "too large" for some users. A disadvantage of this keyboard also is that it takes quite a lot of space on the screen.

How To Incorporate Prediction

The best way how to incorporate prediction in this keyboard would probably be to assign one of the tiles to display predicted words, which could be selected in the same way as any other keys, i.e.by selecting the "direction" where the word is placed.

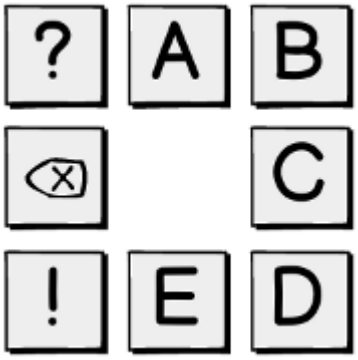


Figure 4.8: 3x3 keyboard — second level

4.3.5 Move-Controlled Keyboard

Fitts' Law, mentioned in section 4.1.1, teaches us that items that are placed on the edges of the screen are those that are most easily reachable. The *Move-Controlled Keyboard* layout follows this rule precisely by placing the keys *only* along the edges.

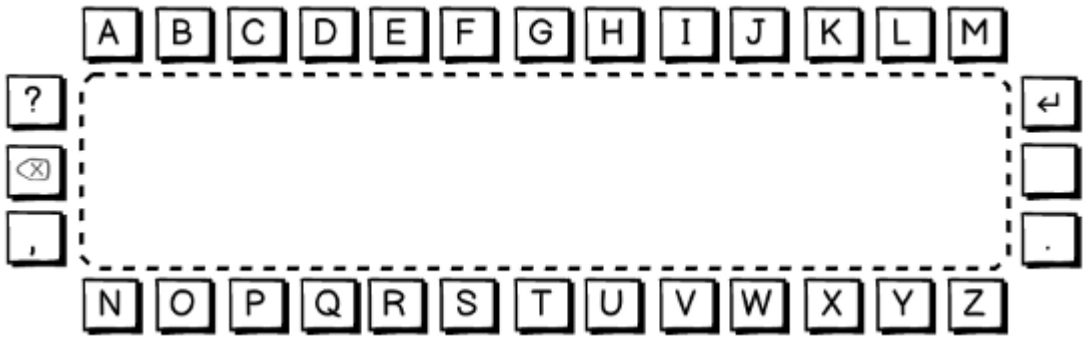


Figure 4.9: Move-controlled keyboard

How It Works

See the figure 4.9 for an illustration of how the *Move-Controlled Keyboard* looks like. There are keys arranged along the edges of the interface, and a large "empty" area in the middle. Actually, this area is not empty — it serves as a "confirmation area" and a resting place for the pointer. The keyboard is controlled solely by moving the pointer, no clicks or presses are needed. If the user wants to write a letter (or select any other key), he has to move the pointer from the middle area to the desired key, and then straight back to middle. The key that is to be selected has to be the last one the pointer points at before it is moved to the middle. That means that if the user moves the pointer from one key to another without

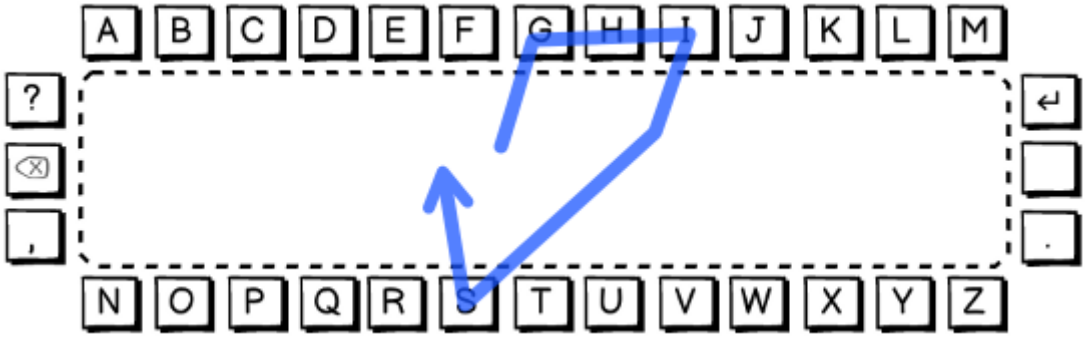


Figure 4.10: Move-controlled keyboard — example of writing the word "is"

returning back to the middle confirmation area, no key is selected. The figure 4.10 shows an example of the pointer's path that will lead to writing the word "is".

The keys have some empty space between them in order to provide the user with "escape points" to return to the middle without selecting any key (the size of these spaces should be configurable).

There is another option of how the layout could look like. According to the Fitts' Law, corners of the interface are the easiest points to reach. In the design proposed above, those corners are not utilized (except that they serve as "escape points"). The possibility to place some keys in the corners suggests itself. Those four keys should contain frequently used actions. For an illustration of this second variant of the *Move-Controlled Keyboard* design, see figure 4.11.

Figure 4.11: Move-controlled keyboard — design with keys in the corners

Suitable Devices

The *Move-Controlled Keyboard* is designed to be used with continuous input devices only. That means everything that is able to move the pointer on the screen can be used to control

this keyboard. No buttons are needed.

Pros and Cons

This keyboard is ideal for users that are unable to perform clicks or presses. With the keyboard types that require pressing a button to select a key, this disability has to be solved by using some kind of auto-click (auto-confirm) feature. With this keyboard, nothing like that is necessary. The user does not have to wait until some timeout expires, and can therefore type faster. A disadvantage for some users may be that this keyboard requires a bit higher level of precision of the pointer movement, because it is necessary that the user moves the pointer from the selected key straight to the confirmation area without touching any other key. It is therefore important that the size and number of the keys is adjustable, so that the user can choose parameters that will suit him and prevent accidental selection of keys. Similarly to the *3x3 Keyboard*, the *Move-Controlled Keyboard* is quite space-consuming. However, allowing to decrease the number of keys in the keyboard configuration can help to make the keyboard smaller, if necessary.

How To Incorporate Prediction

Prediction would be quite easy to incorporate in this keyboard, and would not require any major modification. The predicted words could be displayed as special keys, and could be written by selecting them in the same way as other keys are selected. If the variant of the layout with keys in the corners would be used, the predicted words could be displayed on the bigger keys in the corners, which would make them easy to reach.

4.3.6 Bisection Keyboard

The *Bisection Keyboard* is a bit similar to the *Shifting Keyboard*, described in section 4.3.3. It consists of a single row of keys as well (see figure 4.12). What differs is the way how the keyboard is controlled.

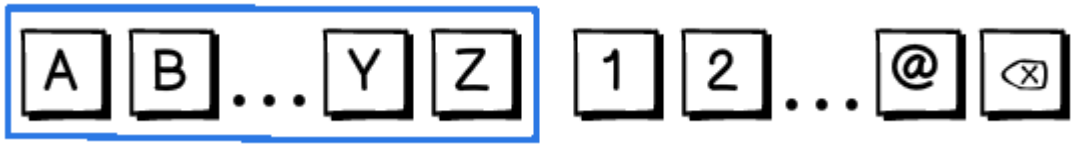


Figure 4.12: Bisection keyboard

How It Works

Controlling the keyboard is based on the idea of interval bisection, known from mathematics. Similar principle is used in informatics for binary search in an ordered list. This method finds a given value by successively halving a discrete interval. Every time the interval is halved,

the half where the key is supposed to lie is chosen and used in the next iteration. The keyboard works similarly, with the difference that it is the user who chooses the intervals.

The keyboard is split into two halves (two intervals). The keys that are at the beginning and the end of each interval are visible, the rest is hidden (which is expressed by three dots — see figure 4.12). Using two navigation buttons (*left* and *right*), the user chooses the half where the key that he wants to reach lies. If the user does that, the interval is expanded to take up the whole screen, and is halved again. The user then repeats this process until one (if the number of keys is odd) or each (if the number of keys is even) of the intervals contains only one key. The last choice of the left or right interval then selects the key. In the end, the keyboard returns to the initial state.

Note that the number of steps to select a key is always the same for all keys (with the exception that for some keys it can be one step less if there are odd-sized intervals), and depends on the total number of keys. Figure 4.13 shows an example of the flow of writing the letter "P" on a keyboard with 52 keys.

A question arises — how to allow the user to get back in case he selected a wrong interval? This can either be done automatically (after some defined period of time elapses and no interval is selected), or there could be a special user action for that, assigned to a button on the input device.

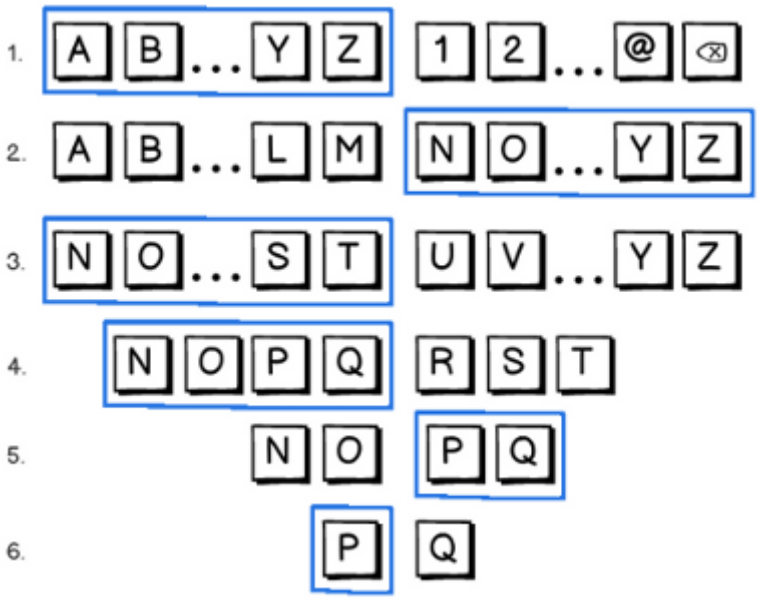


Figure 4.13: Bisection keyboard – example of writing the letter "P"

Suitable Devices

This keyboard can be used either with continuous or discrete input devices. It is possible to select an interval in two steps with a continuous input device — by pointing and then clicking on the desired half of the keyboard. A discrete input device is able to select the interval in

one step, if it has at least two buttons (two buttons can select the intervals directly, without the need to "highlight" them first — one button is assigned to select the left interval, the other to select the right interval). However, the keyboard can work with only one button as well. In such case, an auto-navigation feature has to be used, which would alternately highlight the intervals. The button can then be used to select the currently highlighted interval. If the device provides three buttons, and the user is able to use them all, the third button can be mapped to the "return" action. If no such button is available, the returning must be solved by using a timeout.

Pros and Cons

An advantage of this keyboard is that only one, two, or three buttons are needed to control it. It is also very simple and does not take much space on the screen. A disadvantage is that the user has to know the order of the keys to be able to select the "right" interval. Especially at the beginning of the selection process, it happens that the key the user wants to select is not visible, so the user has to know, or guess, where the key is. What is more, it may not always be possible to order the keys in a straightforward way to help the user determine in which interval the key lies — it is easy to do it for letters or digits, but how to order for example symbols such as punctuation marks? The user will therefore be forced to remember the order of such keys. That means that the keyboard will be harder to use for novice users. This keyboard should not be used with large sets of keys, not only because a large set of keys is harder to order, but also because the number of interval selections needed to get to a single key depends on the total number of keys. Using too many keys would therefore not only increase the number of steps, but also the chance that the user selects wrong interval on his path to the desired key (because of not guessing the position of the key right).

How To Incorporate Prediction

Figure 4.14: Bisection keyboard with two predicted words

This keyboard is probably the hardest to incorporate the prediction in from all the keyboards described in this chapter. It is because this keyboard is based on ordering the keys, and requires that the user knows "where to go" (which interval to select). However, the prediction system generates the words dynamically, so the user can hardly know what words will be predicted. The only option therefore is to always place the predicted words at the beginning and/or end of one of the initial intervals, as those are the positions that can always be seen from the initial state. That limits the number of the words that can be displayed (with the keyboard layout shown in figure 4.12, the optimal number of predicted words would probably be just about two, one at the very beginning and one at the very end

of the keyboard — see figure 4.14). The fewer predicted words can be displayed, the better prediction system has to be used, because it has to predict the words very precisely.

4.3.7 Multi-Level Keyboard

The *Multi-Level Keyboard* is based on a similar principle of "keys inside keys" as the *3x3 Keyboard*, described in section 4.3.4. This keyboard is more general — it allows any number of keys and levels. The tiles are arranged in a grid, and the set of keys that can be reached by selecting a tile is displayed on that tile. Figure 4.15 shows an example of two-level keyboard with six tiles, figure 4.16 shows three-level keyboard with four tiles.

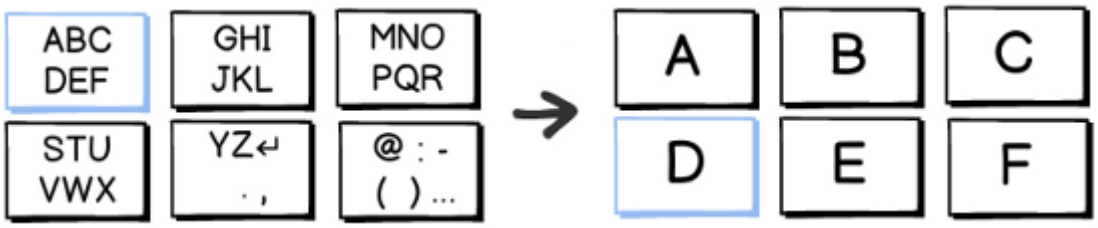


Figure 4.15: Two-level keyboard with six tiles

How It Works

Each tile contains a part of the whole set of keys. If a tile is selected, the keys that it contains are displayed in the next level of the keyboard. If the number of keys is higher than the number of tiles, the keys have to be further split between the tiles, forming another level of the keyboard. In the lowest level, each tile contains only one key. Figure 4.16 shows an example of writing the letter "D" with a layout that has four tiles.

The number of levels depends on the total number of keys and the number of tiles. If there is low number of tiles in the layout, it allows the user to choose between them faster. On the other hand, low number of tiles in combination with a large set of keys will cause that the keyboard has multiple levels, and the user will therefore have to do more steps to reach a key.

Suitable Devices

This keyboard can be used with basically any input device. In case of discrete input devices, pressing the buttons can either cause the tiles to be selected directly (i.e., each button would be mapped to one tile), or there can be one "select" button and several "navigation" buttons that move the cursor from one tile to another.

Pros and Cons

The *Multi-Level Keyboard* is similar to the *Grid Keyboard* (see section 4.3.1), it just adds one more "dimension" to the layout. It shares the advantage of great flexibility with the *Grid*

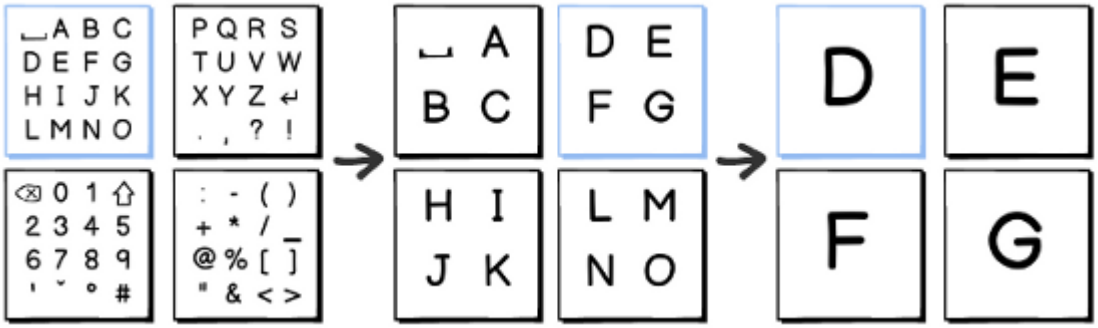


Figure 4.16: Three-level keyboard with four tiles

Keyboard — the keyboard can be configured to display basically any set of keys distributed between any number of tiles. That allows to adjust it to the concrete input device and its user. In case the number of levels is high, a disadvantage is that a lot of button presses are needed to select a key. The number of tiles therefore should be well-balanced with the total number of keys.

How To Incorporate Prediction

Prediction could be incorporated in a similar way as for the *3x3 Keyboard*, i.e. one tile would contain the predicted words. The predicted words could also be distributed between the tiles, such that each tile would contain for example one predicted word. The words could be assigned to the tiles either at random, or based the letters the tile contains. The latter option means that if the tile contains the same letter that the user would have to be type next to write the predicted word, this word would appear on that tile. This principle is similar to one the proposal of displaying predicted words right on the keys that was described for the *Grid Keyboard* in section 4.3.1.

4.3.8 Phone Keyboard

The *Phone Keyboard*, as its name suggests, is based on the design of keyboards used on cell phones. A traditional keyboard, as we know it from PC, would not fit in the limited space cell phones have for it, so it had to be modified to have more than one letter on each key. The software keyboard of this type, designed in this thesis for the EasyControl application, looks very similar (see figure 4.17). However, this keyboard would not be of much benefit in comparison to the *Grid Keyboard* (section 4.3.1) or the *Multi-Level Keyboard* (section 4.3.7), if it just allowed the basic controlling, i.e. that the letters would be written by pressing a key multiple times, depending on the position of the letter on the key. Implementation of this keyboard can of course contain this option, but the keyboard's main usage should be with the prediction system on.

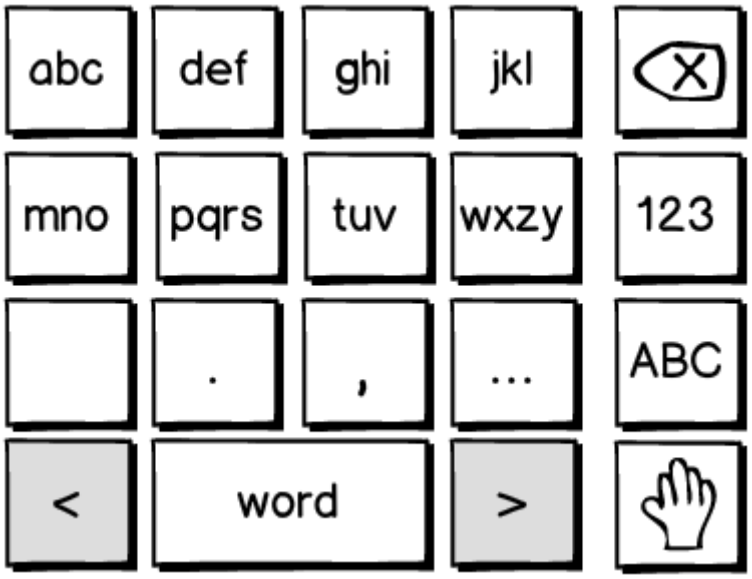


Figure 4.17: Phone keyboard

How It Works

The prediction system for this keyboard is similar to the T9 system known from most cell phones. The user presses the keys only once, no matter which letter on the key he wants to type. The system than looks up a table of words that correspond to the combination of keys entered by the user, and offers a list of these words, among which the user then chooses the desired word. The keyboard therefore has to have some additional keys that would allow the user to "browse" the words and select the right one (see the bottom row of keys in the figure 4.17). Also, there should be a fallback if the word the user wants to write is not in the list. In such case, the keyboard should allow switching to a "manual entry" mode, where the letters can be written individually (key in the bottom-right corner in figure 4.17 illustrates this).

Suitable Devices

The *Phone Keyboard* can be controlled similarly as the other keyboards that are based on a grid of keys. It is possible to use discrete as well as continuous input devices. The number of buttons needed is the same as for the *Grid Keyboard*, described in section 4.3.1.

Pros and Cons

This keyboard has the advantage that the design takes the usage of prediction system into account right from the beginning. Without using it, writing on this keyboard is significantly slower, because it is typically necessary to press the same key multiple times to write a letter. However, the usage of the prediction system can become a disadvantage as well, because it is very important that the prediction system predicts the words sufficiently well. If the word

that the user wants to write does not appear among the first two or three predicted words, the advantage of not having to press the letter keys multiple times will be lost, because the user will have to browse the list of predicted words, which requires pressing keys as well. Among the other advantages, we can mention the possibility to use many types of input devices, and also the small size of the keyboard layout.

How To Incorporate Prediction

Prediction is already an organic part of this keyboard, so there is no need to modify the keyboard to include prediction. The only work that can therefore be done in this area is to improve the prediction system and the mechanism of how the words are selected, to achieve as good match between what the user wanted to write and what was the first predicted word as possible.

4.3.9 Draw Keyboard

Last design presented in this chapter is a keyboard that allows users to "draw" characters instead of having to select them. However, it is important to make the keyboard controllable also with discrete input devices, not just with the continuous input ones (which are usually used with these types of text entry systems). In chapter 3, application called EdgeWrite was introduced (see section 3.2.8), which provides very similar functionality. Since the application is available for developers as a standalone library, it could be utilized for the purposes of this thesis, as there is no reason to re-invent what is already done.

How It Works

The EdgeWrite application consists of a simple square area, which defines the space in which all strokes occur. The strokes are made along the edges and into corners (see figure 4.18). The corners determine the stroke being made — the order in which the corners are entered is what matters. When a stroke is finished, the user "confirms" the stroke, which causes the character to be produced. How the confirmation is done depends on the input device. If the input device is a joystick, confirmation can be done by returning the joystick to the default position. With some other devices, confirming the stroke can be done after some specified time during which the device keeps still.

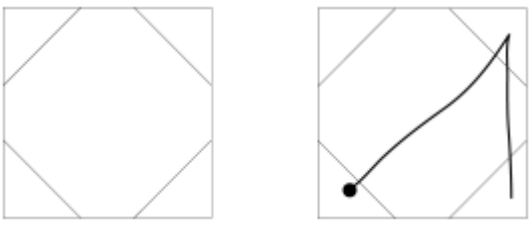


Figure 4.18: EdgeWrite keyboard — the square area (on the left) and an example of writing the letter "a" (on the right)

There is a special alphabet that the user needs to learn to be able write with EdgeWrite. However, it was designed by the author in order to be simple and easily learnable [34]. A part of the alphabet is shown in figure 3.16.

Suitable Devices

This keyboard can be used either with continuous input devices, or with any discrete input device that has at least four buttons.

Pros and Cons

According to the evaluation done by John Wobbrock, the author of EdgeWrite, for his paper [34], an advantage of this keyboard is that typing with it is significantly faster than typing with the selection-based keyboards (a representative of such keyboard is the *Grid Keyboard*). A disadvantage for some users can be that there is a special alphabet that needs to be learned. That also makes the keyboard unsuitable for one-time users.

How To Incorporate Prediction

The EdgeWrite application already contains a word-completion system, as described in section 3.2.8.

Chapter 5

Implementation

Five of the nine keyboard designs presented in section 4.3 of the previous chapter were chosen to be implemented, and are therefore part of the resulting Keyboard application. This chapter discusses the actual implementation of the application. Requirements for the Keyboard application are mentioned at the first place. Then both the application's architecture and functionality is described. Even though a brief overview of the code is provided, this chapter is not meant to serve as the project's documentation. The complete documentation of source codes is available on the CD attached to this thesis. There is also a user manual for the application, which can be found in appendix A.

5.1 Requirements

Requirements for the application are based on the usability guidelines presented in chapter 4, analysis of the target users, and outcomes of discussions with Petr Novák, the author of EasyControl application. Some ideas were also suggested by the people who work with handicapped people in the Jedlička Institute in Prague. An overview of the most important requirements is presented here.

Requirements Related to EasyControl

- The keyboard must be a standalone component (software module), so that it could be used by various application that EasyControl offers (text editor, online chat, e-mail client, etc.).
- The code of the application has to be embedded directly in the supplied library, that will be then used by the EasyControl application.
- The Keyboard application has to offer several types of keyboards, so that there is at least one for each special input device available for EasyControl. Mouse-controlled keyboards should also be available.
- The keyboard has to be able to display actions that are needed to control the application it is attached to (e.g., it should contain a "Send" key if used together with the

Mail application). These actions will not be known before the keyboard is actually run, and they can change with the state of the application.

- The keyboard will not receive input from the hardware devices directly. It will rather receive commands (actions) from the EasyControl application. These commands tell the keyboard what to do. There is a defined set of these commands (they include for example navigation commands, such as “Right”, “Down”, or “Next”).

Other Requirements

- The keys, of which the keyboard consists, must be configurable, i.e. it must be possible to change the arrangement of keys, or even use a different set of keys.
- All colours used in the keyboard should be configurable. This implies it should be possible to create high or low contrast interfaces.
- The keys have to be resizeable. The text on the keys has to be resizeable as well.
- It has to be possible to change the thickness of the border of a key. Border of a selected key should be configurable as well.
- If any time intervals are used for actions, these intervals must be configurable.
- It has to be possible to display an icon on a key instead of text.
- It should be possible to play a sound if a key is pressed.

5.2 Technologies

The main choices of technologies for the keyboard application were more or less determined by the parent application, EasyControl. The EasyControl software is written for the .NET framework, which implies that it is targeted to Microsoft Windows platform. It is written in the C# programming language. Even though it is possible to integrate applications written in different .NET programming languages together, C# was chosen for the keyboard application as well, as it is a modern object-oriented language, and there is no reason why any of the other .NET languages should be preferred to it. The Keyboard application is written for .NET framework version 4.0.

The WPF (Windows Presentation Foundation) technology is used for development of the graphical user interface of the Keyboard application. The WPF replaces an older technology, Windows Forms, and contains a long list of features that make programming the user interface easier and more flexible. User interfaces in WPF are typically declared in a markup language, XAML (eXtensible Application Markup Language). It is an XML-based language used for declaring a graph of .NET objects, and configuring those objects with property values and event handling methods.

5.3 Architecture

5.3.1 MVVM Design Pattern

Architecture of the keyboard application is based on the *Model-View-ViewModel* (MVVM) design pattern. The pattern proposes a way how to separate the user interface (View) from the data and business logic of the application (Model), and represents a standard architecture of today's WPF applications [29].

MVVM architecture consists of three layers — the Model, the View, and the ViewModel. Figure 5.1 illustrates the direction of the dependencies between these three layers.

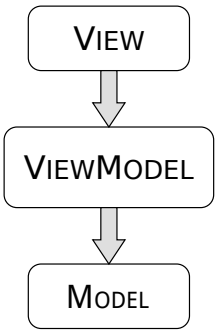


Figure 5.1: MVVM layers

The *Model* contains the application data and the business logic. The *View* contains everything that the user can actually see. It presents the data to the user and allows the user to modify the state of the application via GUI[1] controls.

The *ViewModel* is the "model of a view" — it represents an abstraction of the user interface. As figure 5.1 shows, the ViewModel should not have any knowledge about the View. Since there is no reference from the ViewModel to the View, the View can be easily replaced with a different one, without a need to modify the rest of the application. The ViewModel exposes properties to which Views are "bound". In order to maintain an up-to-date user interface, the View can get notified about the changes in these properties by means of events (generated by the ViewModel). The ViewModel acquires everything it needs from the Model, decorating the data and operations with interfaces that the View can understand and utilize [13]. The Model does not have to know anything about the ViewModel or the View, nor does the View need to know about the Model. There is therefore a loose coupling between the three layers.

5.3.2 The Application

The UML class diagram of the Keyboard application is shown in figure 5.2. The diagram shows five components of the application — the View, the ViewModel, the Model, the Utilities, and the Configuration.

[1] Graphical User Interface

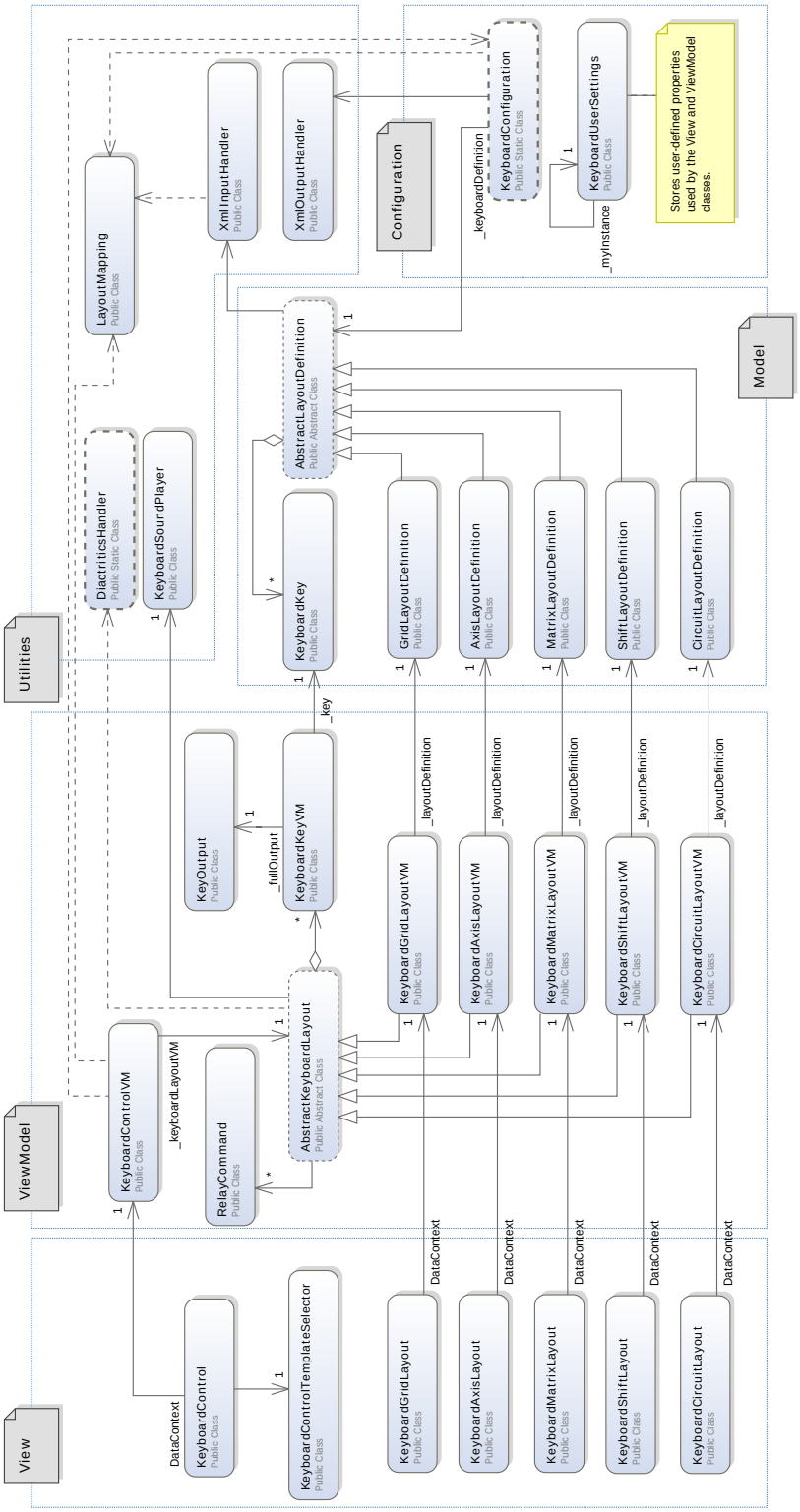


Figure 5.2: UML class diagram of the main parts of the Keyboard application

Members of the classes are not shown in the diagram in order to keep it compact and easier to read. Some classes and associations were also omitted from the diagram for the same reason. Those classes include various converters used by the View (for converting background colours, border styles, etc.), template selectors for keyboard keys, the `Logger` class, and classes needed for a configuration dialogue for various settings that can be adjusted by the user (colors, font, etc.).

The code is organized in namespaces; there is one for each of the five application components. In the rest of this section, the most important parts of these five namespaces will be briefly described. All other details can be found in the project's documentation that is attached to this thesis.

Model

The Model component (`Keyboard.Model` namespace) contains representation of the "data" — the definitions of the various keyboards, and a key. Since the Keyboard application does not require a lot of business logic, and is rather focused on the graphical user interface, the Model component is a relatively small component. A keyboard definition (represented by the `AbstractLayoutDefinition` class and its subclasses) consists mainly of a collection of keys (`KeyboardKey` objects), and optionally of a definition of the keyboard's size (for example the number of rows and columns). Keyboard definitions are stored in XML files, from which they have to be loaded. This is the job of the `XmlInputHandler` that is part of the Utilities component. In the language of the MVVM design pattern, however, it is part of the Model layer. The `XmlInputHandler` knows how to parse the XML file and retrieve the necessary data from it.

ViewModel

The ViewModel component (`Keyboard.ViewModel` namespace) contains classes that prepare the data from the Model component for the usage in View. All the commands and methods for handling "key presses" reside here. An "abstract" keyboard, i.e. a keyboard without any actual graphical user interface, is represented by the `AbstractKeyboardLayout` class and its subclasses. There is one subclass for each of the keyboard designs, and each of these subclasses refers to the corresponding keyboard definition from the Model. The most important part of an "abstract" keyboard is a collection of "abstract' keys, represented by `KeyboardKeyVM` object.

Which type of keyboard will be presented to the user is a matter of configuration. A common component that can be used by client applications to place the keyboard somewhere in their own View is therefore offered, rather than requiring the clients to pick the control for a concrete keyboard type. That allows for greater flexibility — if a new keyboard type is added to the Keyboard application, the client (parent) application does not have to change anything to use this new keyboard. `KeyboardControlVM` is an abstraction of this component. The diagram in figure 5.2 shows that the `KeyboardControlVM` holds a reference to the `AbstractKeyboardLayout`, it is therefore not dependent on any concrete implementation (subclass) of this class.

View

The View (`Keyboard.View` namespace) contains everything that is needed to actually display the keyboard to the user.

The code for the View of each keyboard type is written completely in XAML. There is a resource dictionary for each keyboard, that includes styles and templates for its GUI components. Templates are heavily used in the Keyboard application, because they allow to completely redefine the look and feel of any GUI component, and are reusable. More importantly, templates are utilized by template selectors that can be used to define different appearance for controls of the same type. This is needed for example to display keys, obtained via binding to the collection of keys from ViewModel, in different colours, based on the type of the key (digits can have different colour than letters, etc.).

The Keyboard control that is to be used by the client applications is represented by the `KeyboardControl` class. This component receives input from the client application (for example, actions from the input devices), and sends out output from the keyboard (for example, a letter to be typed). The `KeyboardControl` offers several events the client application can register handlers to, and can therefore get notified each time the event occurs. These events are three — `OnChars` (sending a textual output), `OnAction` (sending an action, for example "Delete"), and `OnExtraKey` (sending an output from "extra" keys that contain actions supplied by the client application). Again, using events allows for better flexibility — the client application can choose not to register a handler for an event, which only results in that it will not receive any notification about that event. That means that in case the Keyboard application will be improved in the future by adding more events, the client application can use the newer version without the need to change its own code. The `KeyboardControl` component uses a template selector (`KeyboardControlTemplateSelector`) to determine *how* to display a concrete keyboard design. The correct View class is chosen based on the configured type of the keyboard.

Utilities

The `Keyboard.Utilities` namespace contains classes from the MVVM's Model layer that are separated in their own namespace to achieve better organization of the code. There are two classes for handling the XML input and output, `XmlInputHandler` and `XmlOutputHandler`. The first one is used to retrieve data from the XML files with keyboard definitions and configuration, the other one is used to write back to these files, or create new ones. Then there is a utility class `DiacriticHandler`, used to combine diacritical marks and letters into letters with diacritic. A `KeyboardSoundPlayer` object can be used to play sounds in the keyboard. The `LayoutMapping` class is something like a "converter" that knows how to determine the keyboard type from the value from configuration (for example, from a name). It contains two factory methods that are used to obtain instances of `AbstractKeyboardDefinition` (needed by `XmlInputHandler`) and `AbstractKeyboardLayout` (needed by `KeyboardControlVM`). The last class in this namespace is the `Logger`, used by all classes to log various events and errors that may occur in runtime.

Configuration

The last component, Configuration (`Keyboard.Configuration` namespace), takes care about everything related to the keyboard's configuration and customization. The `KeyboardUserSettings` class holds various user-defined properties needed by the View and the ViewModel components. These properties can be set by the user using a configuration dialogue. If the application is closed, the configured properties are not lost; they are stored in the user's profile (`ProfileKeyboard`). The keyboard therefore looks and behaves exactly the same the next time the user runs it. The other important class in this namespace is the `KeyboardConfiguration` class. This class maintains the Keyboard application's configuration directory. It can be used to retrieve a list of available keyboards (which are defined by means of XML files). This list appears in the configuration dialogue, where the user can choose which of the keyboards will be displayed in the View.

5.4 Integration in EasyControl

The Keyboard application was originally developed as a standalone library (DLL) that could be linked to any client application that wanted to use it. However, the author of the EasyControl application, to which the keyboard is mainly targeted, wanted to have the Keyboard application integrated directly in his own library. The Keyboard's code therefore had to be copied to an assembly called `InputChannelAppBase` that is part of the EasyControl project. The `InputChannelAppBase` is the main library used by all applications developed for EasyControl, this change will therefore allow the author to use the keyboards directly in any of these applications. Nothing had been changed otherwise, so there should not be any difference in using the keyboard directly integrated into `InputChannelAppBase` library, or as a standalone, dynamically linked library.

Also, there was a requirement from the author of EasyControl (who is the supervisor of this thesis), that the keyboard has to accept input actions produced by his application. That means the Keyboard depends on the EasyControl's API[2] in this particular case (however, this is the only one).

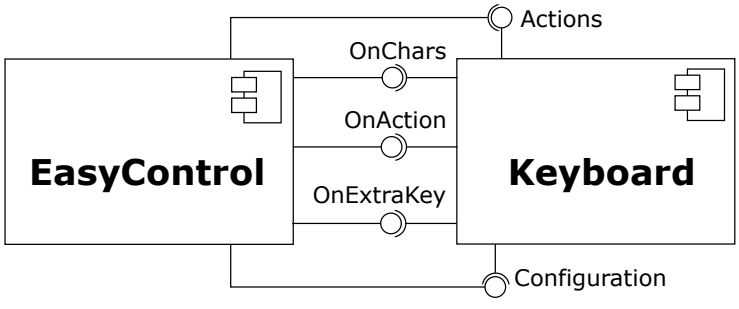


Figure 5.3: Connection of the Keyboard and EasyControl

[2]Application Programming Interface

Figure 5.3 shows the important interfaces that the Keyboard component provides and requires. The input actions, represented by the `Actions` enumeration, are the only required interface. As already mentioned earlier, the Keyboard provides three events, representing the keyboard's output. The keyboard's user profile, that is part of the global user profile of the EasyControl application, and from which various keyboard settings are read (see section 5.7.3), is mainly used to configure the Keyboard application. However, the keyboard can also be configured during runtime. The client application can offer special actions that need to be displayed on the keyboard, but that are not "known" in advance (i.e., when the keyboard layout is designed), or that change with the state of the client application. These actions can be passed to the keyboard through the `Configuration` interface. If there are keys reserved for these actions on the keyboard, these keys are filled with the received actions. That means the number of special actions the keyboard displays is limited by the number of keys that were reserved for them in advance (by the designer of the keyboard layout).

An example of usage of the special actions is an e-mail client application, that consists of a contact list, a list of recipients, and a window to write the message to. If the cursor is on the contact list, actions like "Add to recipients", or "Create new contact" will be probably needed. However, if the user moves the cursor to the window where the message is written, such actions do not make sense any more. Different ones are needed instead, such as "Send", or "Save to concepts". That means that each time the user switches "focus" to another component in the client application, the application should supply the keyboard with a new set of actions that the user may need in the new context (if necessary).

It was mentioned that the keyboard receives input actions (commands) from the EasyControl applications. EasyControl receives signals from the input devices and "converts" them to actions that are then passed to the Keyboard application. These input actions are then further processed by the keyboard. The response of the keyboard then depends on the concrete keyboard type that is used, even though some input actions cause the same response from all keyboard types. A keyboard type can also choose not to respond to some input actions, in case they are not relevant to that particular keyboard type.

There are fifteen actions that can be generated by the EasyControl application. Outputs of the input devices are mapped to these actions. Most of them are navigation actions. Actions for navigation by directions are *Right*, *Left*, *Down*, *Up*, *RightUp*, *LeftUp*, *RightDown*, and *LeftDown*. Actions for "simple navigation" are *NextItem*, *BackItem*, *NextRow*, and *NextColumn*. Then there is *Action*, that typically means confirmation, and *Escape*, which is an equivalent of cancellation. The last action, called *None*, is ignored by the keyboard, since it is not supposed to do anything.

5.5 Implemented Designs

Five types of keyboards were implemented. Each keyboard type is based on one of the designed keyboards introduced in section 4.3. This section will therefore not describe the basic principles of each keyboard again; it will rather focus on details of the functionality, in particular on interpreting the input actions.

5.5.1 Grid Keyboard

The "Grid Keyboard" is a traditional keyboard, whose design was described in section 4.3.1. Figure 5.4 shows how the actual implementation of this design in the Keyboard application looks like. The user can configure the number of rows and columns.

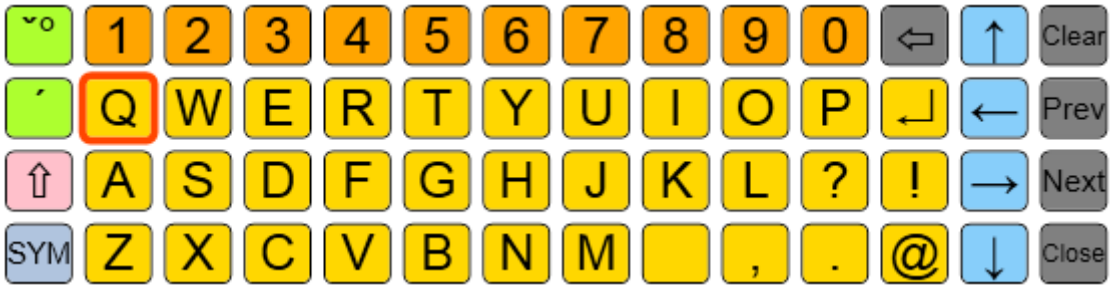


Figure 5.4: Keyboard Application: Grid keyboard, QWERTY layout

There are three possibilities of how to control this keyboard:

- by directional navigation, and confirmation
- by simple navigation, and confirmation
- by moving pointer, and "clicking"

By using navigation actions or moving the pointer, the user selects a key. The keyboard highlights this key with a thick border (in figure 5.4, "Q" is highlighted). The key can then be "pressed" by using the confirmation action or by clicking on it. If a key with diacritic, or the Shift key is pressed, it is highlighted with a different colour and thicker border, so that the user knew it is active.

The input actions from EasyControl are interpreted as follows:

- *Right* — move the cursor (the highlight) by one key to the right. If the end of a row was reached, jump to the first key in that row.
- *Left* — move the cursor by one key to the left. If the beginning of a row has been reached, jump to the last key in that row.
- *Down* — move the cursor by one key down (south). If the end of a column has been reached, jump to the first key in that column.
- *Up* — move the cursor by one key up (north). If the beginning of a column has been reached, jump to the last key in that column.
- *RightUp* — move the cursor diagonally to the next key in the north-east direction.
- *LeftUp* — move the cursor diagonally to the next key in the north-west direction.
- *RightDown* — move the cursor diagonally to the next key in the south-east direction.
- *LeftDown* — move the cursor diagonally to the next key in the south-west direction.
- *NextItem* — move cursor to the next key on the right. If the end of a row has been reached, jump to the first key of the next row. If the end of the last row has been reached, jump to the first key of the first row.

- *BackItem* — move cursor to the next key on the left. If the beginning of a row has been reached, jump to the last key of the previous row. If the beginning of the first row has been reached, jump to the last key of the last row.
- *NextRow* — same as *Down*.
- *NextColumn* — same as *Right*.
- *Action* — "press" (confirm) currently highlighted key.
- *Escape* — N/A (there is nothing to cancel).

5.5.2 Keyboard with Coordinates

Design of the "Keyboard with Coordinates" was described in section 4.3.2. The user interface of this keyboard type in the Keyboard application is shown in figure 5.5. There is also an option to hide the row and column numbers in the keyboard's configuration. Showing the numbers is important if the keyboard is controlled by directly entering the coordinates, otherwise it is not necessary. The user can configure the size of the keyboard (i.e., the number of rows and columns).

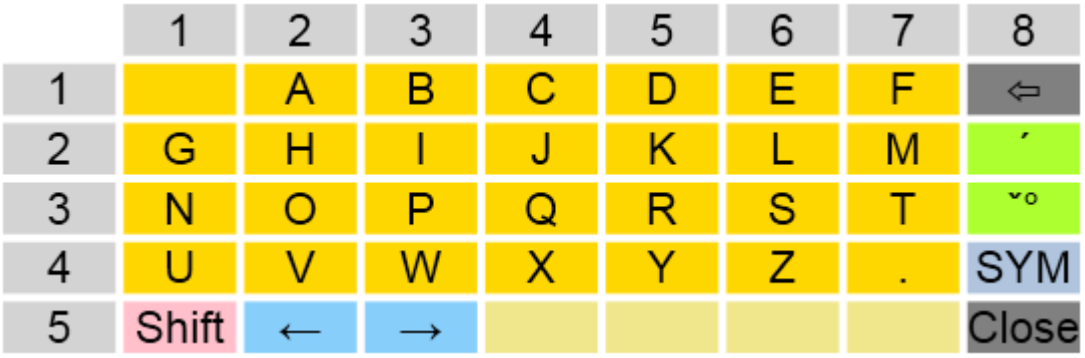


Figure 5.5: Keyboard Application: Keyboard with coordinates

There are two ways how to control the keyboard:

- by directly entering the coordinates
- by "simple navigation" and confirmation

Since the EasyControl application cannot send the numbers directly, there must be a mapping between the directional navigation actions and the numbers. This mapping is needed for the first way of controlling the keyboard. If the user controls the keyboard like this, no confirmation action is needed — selection of a corresponding row and column is done immediately when a number is pressed. When first coordinate is entered, the corresponding row is highlighted in the GUI. When the second coordinate is entered, the key in that column is pressed. If the user made a mistake and selected a wrong row, he can undo the selection by using the cancel action. It is also possible to turn on the automatic cancel, which discards the selection after a specified time interval. Note that if the size of one of the keyboard's dimensions is be greater than eight, it will not be possible to directly enter the

higher coordinate numbers, and therefore it will be more appropriate to control the keyboard using the simple navigation.

The other option allows the user to move the selection of row and column. Simple navigation actions are used for this. Using the *NextItem* and *BackItem* actions, the user can first choose a row — the row is highlighted in the GUI, and until the user confirms the selection using the confirmation action, he can move the selection to another row. When he confirms the selected row, he can then choose the column similarly. When he confirms the selection of the column, the key on the intersection of the selected row and column is pressed. The row selection can be cancelled by using the *Escape* action, or by turning on the automatic cancel. It is also possible to move the selection of row and column "at once" — the *NextRow* and *NextColumn* actions are used for that. The *NextRow* action moves the row selection, while the *NextColumn* action moves the column selection. In this case, the user does not have to confirm the row selection individually (he can even start with selecting the column first), it is enough to confirm the final selection.

If a key with diacritic or the Shift key is selected, an indication that a special key is active is displayed in the top-left corner of the keyboard.

Here there is a summary of how the input actions from EasyControl are interpreted by this keyboard:

- *Right* — coordinate 5
- *Left* — coordinate 4
- *Down* — coordinate 2
- *Up* — coordinate 7
- *RightUp* — coordinate 8
- *LeftUp* — coordinate 6
- *RightDown* — coordinate 3
- *LeftDown* — coordinate 1
- *NextItem* — highlight the next row/column. If the last row/column has been reached, jump to the first row/column again. Rows are highlighted from left to right, columns from top to bottom.
- *BackItem* — highlight the previous row/column. If the first row/column has been reached, jump to the last row/column.
- *NextRow* — highlight the next row. If the last row was reached, jump to the first row again. Rows are highlighted from left to right.
- *NextColumn* — highlight the next column. If the last column has been reached, jump to the first column again. Columns are highlighted from top to bottom.
- *Action* — confirm the selection.
- *Escape* — cancel the selection.

5.5.3 Shifting Keyboard

The "Shifting Keyboard" is the simplest in both how it looks like and how it is controlled. Its design was described in section 4.3.3. The actual implemented user interface is shown in figure 5.6. Only a part of the whole row of keys is shown to the user. It is possible to configure how many keys will be visible.

Figure 5.6: Keyboard Application: Shifting keyboard

This keyboard can be controlled either with the directional navigation or the simple navigation. Confirmation is needed in both cases. The user can move the "window" that highlights the selected key to the left or right. By using the confirmation action, the key that is highlighted can be pressed. The window always stays in the middle (unless there is not enough keys to the right or left), so it looks like it is the row of keys that is shifting (thus the name of the keyboard).

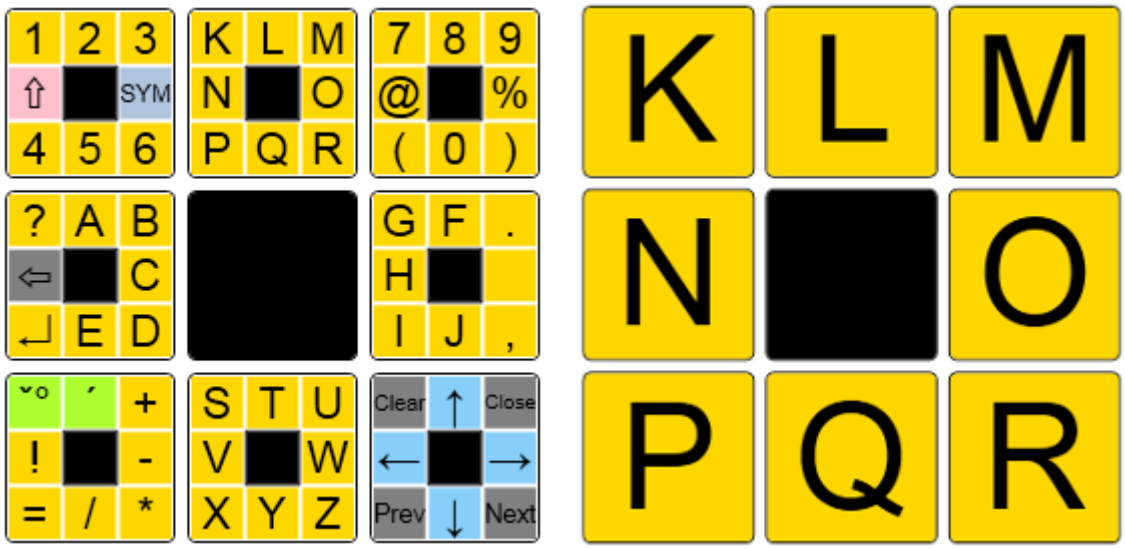
Even though the keyboard offers advanced navigation options, for example jumping to the first or last key of the keyboard, these actions do not have to be used by the users. It is expected that this keyboard will be controlled by users that cannot control more complicated interfaces of the other keyboard types — such users will probably not be able to handle more than about three actions anyway.

The input actions are handled by the keyboard as follows:

- *Right* — move the window by one key to the right. If the last key of the keyboard has been reached, continue from the beginning again.
- *Left* — move the window by one key to the left. If the first key of the keyboard has been reached, jump to the last key.
- *Down* — move the window to the very beginning of the keyboard.
- *Up* — move the window to the very end of the keyboard.
- *RightUp* — move the window by x keys to the right, where x is the number of visible keys. If the end of the keyboard has been reached, continue from the beginning.
- *LeftUp* — move the window by x keys to the left, where x is the number of visible keys. If the beginning of the keyboard has been reached, continue by jumping to the last key of the keyboard.
- *RightDown* — N/A
- *LeftDown* — N/A
- *NextItem* — same as *Right*.
- *BackItem* — same as *Left*.
- *NextRow* — N/A
- *NextColumn* — N/A
- *Action* — confirm the key that is currently in the window.
- *Escape* — N/A

5.5.4 3x3 Keyboard

The design of the "3x3 Keyboard" was described in section 4.3.4. The keyboard has two "levels". In the first level, the keys are distributed among eight "tiles" that are arranged in a 3x3 grid (see figure 5.7(a)). The middle tile is always empty. By selecting one of the tiles, the keyboard is switched to the second level (see figure 5.7(b)). This level shows only the



(a) Level 1 (b) Level 2

Figure 5.7: Keyboard Application: 3x3 keyboard

eight keys from the selected tile. A key can be selected in the same way how a tile is selected. The keyboard then returns to the first level.

There are three possible ways how to control the keyboard:

- by selecting the cardinal directions directly (no confirmation needed)
- by using simple navigation actions and confirmation
- by moving the pointer and clicking to confirm the selection

The first way is the "recommended" one, since it requires the least steps to select a key. Each tile (in the first level) or key (in the second level) can be selected directly by using the action that is mapped to the cardinal direction where the tile/key is placed. For example, the tile with the keys shown in figure 5.7(b), was selected by using the *Up* action, as this action corresponds to the north direction.

The second way how to control the keyboard is by using the simple navigation. It is possible to navigate through the keyboard by moving the cursor from one tile (or key) to another. The middle (empty) tile/key is always skipped. The confirmation action is then needed to select the desired tile (or key).

The keyboard can also be controlled with devices that can move the pointer. A tile or key is selected by pointing to it and preforming a click.

In case the user accidentally selected a wrong tile, the cancel action can be used to return to the first level of the keyboard. This cancel action can either be mapped to an output of the controlling device, or it is possible to turn on the automatic cancel function, which automatically cancels the tile selection after a specified time.

If a key with diacritic or the Shift key is selected, the middle tile in the first level changes colour, and displays text indicating which key is active.

The following actions can be used together with this keyboard:

- *Right* — select the tile/key in the east direction.
- *Left* — select the tile/key in the west direction.
- *Down* — select the tile/key in the south direction.
- *Up* — select the tile/key in the north direction.
- *RightUp* — select the tile/key in the north-east direction.
- *LeftUp* — select the tile/key in the north-west direction.
- *RightDown* — select the tile/key in the south-east direction.
- *LeftDown* — select the tile/key in the south-west direction.
- *NextItem* — highlight the next tile/key. The direction of the navigation is by rows, from the north-west to the south-east tile/key, and then again from the beginning.
- *BackItem* — highlight the previous tile/key. The order in which the tiles/keys are selected is reversed in comparison with the *NextItem* action.
- *NextRow* — highlight the next tile/key in the current row. If the last tile/key has been reached, jump to the first one in the row.
- *NextColumn* — highlight the next tile/key in the current column. If the last tile/key has been reached, jump to the first one in the column.
- *Action* — confirm the currently highlighted tile/key.
- *Escape* — return to the first level of the keyboard without selecting any key.

5.5.5 Move-Controlled Keyboard

The "Move-Controlled Keyboard" differs from the others in that it is targeted to devices that can move the pointer on screen. The design was described in section 4.3.5. The Keyboard application includes implementation of the simpler layout, without keys in the corners (see figure 5.8).

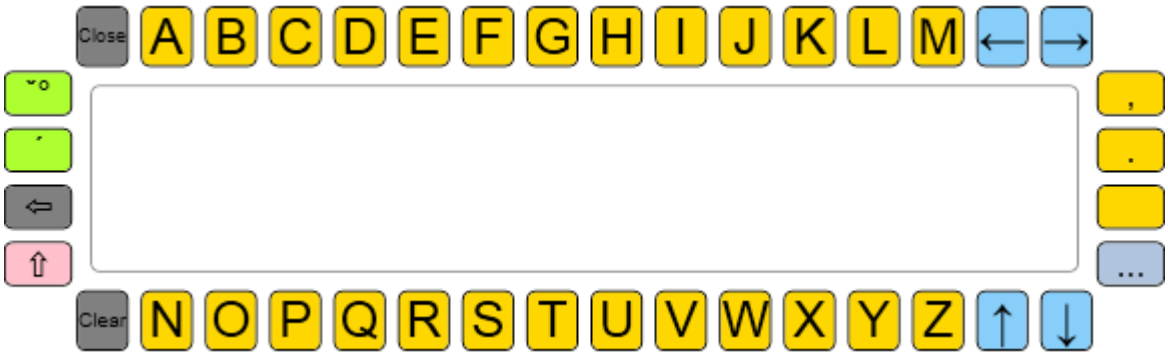


Figure 5.8: Keyboard Application: Move-controlled keyboard

The principle of how keys are selected was already described in section 4.3.5. Basically, a key is typed by moving the pointer to it and then back to the middle, where the confirmation area is situated. It is possible to configure the space between the keys, as well as the space between the confirmation area and the keys. The user can therefore configure the keyboard

according to how precisely he or she is able to move the pointer. It is also possible to configure the number of keys displayed around the confirmation area.

This keyboard is the only one that does not need to receive any input actions from the EasyControl application.

5.6 Common Features

The previous section described the functionality characteristic to each keyboard type. This section describes the functionality of the Keyboard application that is independent on the chosen keyboard type.

5.6.1 Keyboard Output

The Keyboard application provides basically two types of output — characters and actions. Output from the keyboard is sent to the client application (EasyControl), that can decide how to process it. The three events that are used to send the output to the client application were already introduced in section 5.4.

The `OnChars` event is raised every time the keyboard wants to send a textual string to the output. The client application then typically appends this text to the text that is being written, and is displayed somewhere in the GUI.

There are two events for actions. The `OnAction` event is raised when a predefined action needs to be sent to the output. These predefined actions are present on the keyboard in a form of special action keys. The output of those keys are names of the actions ("Delete", "Close", "Clear", and so on), and any text or icon can be defined to be displayed on the keys. The client application receives the name of the action, and if it is a known action, the application processes it (for example, if the "Delete" action is received, deletes the last character from the text that is being written).

There was a requirement that the Keyboard application also needs to be able to display actions that come from the attached (client) application during runtime. There are special keys reserved for these actions. Each keyboard can be configured to have some number of these keys. The keys are initially empty — the Keyboard application fills them with content when it receives a set of "external" actions from the client application. The content of these keys can even change during runtime, depending on the state of the client application. An example of an external action is a "Send" action, present if the client application is an e-mail client. If a key with such action is pressed, the `OnExtraKey` event is raised. The name of the action serves as an identifier of the action, similarly as in case of the predefined actions.

5.6.2 Handling Diacritics

The Keyboard application is targeted to Czech-speaking users, therefore it has to provide a way how to write letters with diacritics ("á", "é", "ě", "ů", "ř", "š", "ť", and others). There are two possible ways how the keyboard can be used to write these characters.

The first way is to include letters with diacritics in the keyboard layout directly, i.e. that they are displayed on the keys, and when the key is pressed, they are sent to the output (for

example, to a text window). This possibility has the advantage that the user sees the letters directly, so he or she does not have to learn a special way how to type them. A disadvantage is that since there are quite a lot of these characters (15 in Czech alphabet), they will take considerable space on the keyboard.

The other way is to have special keys for the diacritical marks. A letter with diacritic (for example, "č") is then typed by pressing a key with the diacritical mark first (in our example, the caron, "ˇ"), and then pressing the key with the letter itself (in our example, "c"). In case the user enters an invalid combination of a diacritical mark and letter, for example a caron ("ˇ") and letter "a", the characters will not be combined, but they will be sent to the output exactly as how they were typed (i.e., "ˇa"). The advantage of this method of typing letters with diacritics is that it saves space on the keyboard (three diacritical marks are used in Czech alphabet, therefore maximum of three keys are needed). On the other hand, the users have to press two keys to write one letter, and they have to remember that they have pressed the special key (the keyboard should offer some indication to eliminate this).

In the Keyboard application, it is possible to use either of these possibilities. It offers special keys for diacritics, but the usage of these is optional. By using a lookup table, the application determines how to combine diacritical marks with letters to produce all possible characters from Czech alphabet. If needed, it would be very easy to extend this table to support more letters with diacritic, used in other languages.

5.6.3 Switching Layouts

For all keyboards there is a possibility to change their set of keys for another one (and back) by selecting special key. There can be any number of these keys in the keyboard. Each one contains a reference to another keyboard layout (see section 5.7.1 for details of how this is realized). This keyboard layout is displayed instead of the current one if the key is selected. That means the keyboard type remains the same, only the set of key changes. It is even possible to change the size of the keyboard (for keyboard types that allow it). However, this should be used very carefully, since it can be confusing to the user. The new layout should contain a key that allows the user to get back to the previous layout (possibly via another layout(s)).

Typical usage of the "layout switching" functionality is when we want to offer more characters than how many fit on the keyboard (even though most of the keyboard types can be configured to contain any number of keys, it is not always wise to try to display everything on one screen). For example, the layout of the "Grid Keyboard" shown in figure 5.4 contains letters and digits, but characters like colon, dash, star, or parentheses are missing. These (and other) characters are part of another layout that is displayed when the blue "SYM" key is pressed.

5.7 Configuration

This section explains how the configurable parts of the Keyboard application are realized[3]. To achieve as much flexibility of the keyboard layouts as possible, they can be defined entirely

[3]More details about how to configure the keyboard, or define a new keyboard layout and add it to the list of available keyboards, can be found in the user manual (appendix A).

by the user. XML files are used to store this definitions. Configuration of available keyboards is done similarly. Various user settings, such as colours, font, or time intervals, are stored in a user profile. This profile is saved in an XML file, which allows to load different profiles for different users of the keyboard.

5.7.1 Layouts

Keyboard *layout* is represented by an ordered set of keys that are displayed on the keyboard. The XML file that defines the keyboard contains a set of tags that define keys, and optionally (depending on the type of keyboard) also some tags that define parameters of the keyboard (e.g., number of rows and columns for the "Grid Keyboard", or number of visible keys for the "Shifting Keyboard"). In the resources of the Keyboard application, XML schema that validates the XML files with keyboard layout definitions is available.

Each key (defined by an XML element `Key`) has nine properties (each represented by a nested element):

- **Text displayed on the key** (`DisplayString`). This is what the user will see on the key. It can be any textual string — from one character to a whole word, or even several words.
- **Icon for the key** (`DisplayImage`). An icon can be displayed on the key instead of text. The file with the image can be specified in this optional element. If no icon is defined, text is displayed on the key.
- **Output of the key** (`Output`). This element specifies the "output" of the key. In case of keys with letters or digits, the output will typically be the same as the displayed text. However, this is not necessarily true for other keys — the output can also be a name of an action (e.g., "Delete"), or, in case of keys that can switch layouts, a name of a file with keyboard definition.
- **Output in "Shift" mode** (`ShiftOutput`). This element is similar to the `Output` element. The only difference is that this element specifies the output of the key if the "Shift" mode is turned on. This element is optional. It is not necessary to specify the `ShiftOutput` for keys with letters — those are automatically converted to upper case if the "Shift" mode is on.
- **Sound for the key** (`Sound`). This is another optional element. It specifies a file with the sound that should be played if the key is pressed.
- **Colour of the key** (`BackgroundColor`). Specifies background colour of the key. If background colour is not set, global settings of colours are used to determine the colour of the key based on its type (see the `Type` element described below).
- **Colour of text** (`TextColor`). Specifies colour of text that is displayed on the key. If this element is not present, the text colour is determined from the global settings.
- **Colour of the output** (`OutputColor`). Defines preferred colour of the key's output. This property is not used by the Keyboard application itself, but can be utilized by

the parent application to display the characters received from the keyboard in different colours (for example, if a key with letter "a" had this property set to red, the letter would always be displayed red in a text editor attached to the keyboard, even if the rest of the text was black). This property is optional.

- **Type of the key** (`Type`). This element specifies which group of keys the key belongs to. There are eleven possible types of keys:
  - **Letter** — letters
  - **Digit** — digits
  - **Symbol** — other characters
  - **Diacritic** — diacritical marks
  - **Shift** — the Shift key
  - **SwitchLayout** — keys that "switch keyboard layouts", i.e. they cause that the set of keys displayed on the keyboard is changed
  - **Delete** — a key to delete already typed characters
  - **Control** — keys like Ctrl, Alt, etc.
  - **Navigation** — navigation keys that move the cursor
  - **Action** — keys for any other actions (e.g. Close, Clear)
  - **Extra** — keys for actions supplied by the parent application

A set of keys is the core of a keyboard layout configuration. However, there are a few more properties that can be configured for some types of keyboards. These properties are related to size of the keyboard. The element that contains them is called `SizeDefinition`. The "3x3 Keyboard" is the only one whose size cannot be configured, and thus its layout definition contains only definitions of keys. Keyboard parameters (each defined inside an XML element) that can be configured for the other keyboards are the following:

- `Rows` — the number of rows the keyboard has. This element can be used when defining a layout for the "Grid Keyboard" or "Keyboard with Coordinates".
- `Columns` — the number of columns the keyboard has. Besides the "Grid Keyboard" and "Keyboard with Coordinates", this element can be also used for the "Shifting Keyboard", for which the content of the element defines the number of visible keys on the keyboard.
- `Horizontal` — this element is used in a definition of the "Move-Controlled Keyboard". It says how many keys the keyboard has in the rows above and below the middle confirmation area.
- `Vertical` — this element is used together with the `Horizontal` element. The `Vertical` element specifies how many keys are there in the columns to the left and right from the middle confirmation area.

5.7.2 Keyboards

The Keyboard application needs to know where to find the keyboards it should offer to the user, so that he or she could choose which one to use. For this purposes, there is a

configuration file, by default called *Keyboards.xml*, that contains a list of available keyboards. Each keyboard is defined by its type, its name, and a file that contains its definition. This file can of course reference other files with layout definitions (in the definitions of keys that switch layouts), but it is not necessary to specify these referenced files in the configuration — it is enough to define the "starting" one for each keyboard.

An entry for a keyboard in the list of keyboards is represented by a `Keyboard` element. This element contains three child elements:

- `Name` — name of the keyboard. It can be anything that helps to identify the keyboard (e.g., "Shifting keyboard with alphabetically ordered keys", "Black & White Keyboard", or "Peter's keyboard").
- `Type` — type of the keyboard. There are five possible values that can be inside this element: `GridKeyboard` (for the "Grid Keyboard"), `AxisKeyboard` (for the "Keyboard with Coordinates"), `ShiftingKeyboard` (for the "Shifting Keyboard"), `MatrixKeyboard` (for the "3x3 Keyboard"), and `CircuitKeyboard` (for the "Move-Controlled Keyboard").
- `LayoutFile` — name of the file that contains the definition of the keyboard's layout. Relative path to the file from the configuration file can be used as well.

5.7.3 User Profile

The EasyControl application supports user profiles, so it was natural to implement them in the Keyboard application as well. Users of the application can have their own profiles, and when the application is started, they can simply choose this profile to make the application look and behave the same as when they run it last time. The application can therefore be shared by multiple users, and it needs to be configured only once for each of them. Some users can also share a common profile, or there can be default profiles for specific groups of users (for example, for beginners, or one-time users).

There are many properties that can be adjusted in the user profile. Everything is configured in the EasyControl application's configuration dialogue, to which the keyboard configuration was integrated.

Properties that are included in a user profile are the following:

- Colours — basically everything that is displayed in the user interface can be configured to have different colour. The options include background colours of keys, text colours, colours of keys' borders, colours of highlighted keys, colour of the confirmation area in the "Move-Controlled Keyboard", colour of the empty middle keys in the "3x3 Keyboard", etc.
- Border thicknesses — it is possible to configure the thickness of keys' borders, highlighted keys' borders, or thickness of the grid in "Keyboard with Coordinates".
- Space between keys — the amount of empty space between keys (keys' margin) can be adjusted as well.

- Font — font family of the font that is used for the texts in the keyboard can be changed. It is not necessary to configure the font size, because the texts are resized automatically to fit the keys.
- Automatic cancel — in the "Keyboard with Coordinates" and the "3x3 keyboard", automatic cancel action is utilized (see sections 5.5.2 and 5.5.4 for explanation). The length of the time interval can be configured, and it is also possible to entirely disable the automatic cancel function.
- Displaying images — it is possible to disable displaying icons on keys. If this option is turned off, no images will be displayed on keys, even if there are images specified for some keys in the keyboard's definition.
- Playing sounds — it is possible to disable (mute) all sounds.

5.8 Future Work

This section provides a brief description of three proposed extensions that could be added to the Keyboard application in the future.

5.8.1 Prediction System

A prototype of a prediction system has already been developed by Martin Vogal [32]. It was intended to be used together with the EasyControl application. The system is able to predict words and sentences based on a dictionary that is created from words that the user has typed in the past. Even though the system implements some interesting features (for example, predicting based on word classes or emotional tone of the sentence being typed), it is unfortunately not in a state that it could be used straight away. The reason is that the prediction system is not prepared to cooperate with another system (for example, it does not offer any interfaces to connect to). Therefore, refactoring of the project's source code would first be needed before the system could be integrated in the Keyboard application.

A disadvantage of Vogal's prediction system is that it does not contain any predefined dictionary. Therefore, the user must use the system for some time before the prediction starts to be useful. Also, the system is not able to eliminate typing errors. Since it does not check the typed words against any dictionary, nothing will prevent it from learning misspelled words.

If a prediction system should be integrated in the Keyboard application, it could be based on Vogal's implementation, but it would be appropriate to add more improvements to it, such as the predefined dictionary. How the predicted words could be displayed in the various keyboard types was already proposed in section 4.3.

5.8.2 Abbreviation Expansion

Another feature that can help the users to speed up their typing is so called abbreviation expansion. Such feature allows to define abbreviations for often used words, phrases, or sentences. The user can then type just the abbreviation instead of all the words it stands

for. For example, “hay” could stand for “How are you?”. The abbreviation of course does not have to consist just of the first letters of the abbreviated words; it can be any other sequence of characters. For example, “hk” can expand to “Hi, my name is Klára.”.

There are several options of how the abbreviation expansion could work. The user could for example type the abbreviation, and then press a special key that would expand it and send the result to the output. The abbreviations could also be expanded automatically. However, there would then have to be mechanism how to “undo” the expansion in case the user actually wanted to type the sequence of characters corresponding to the abbreviation, without expanding it.

An abbreviation expansion system is often included in text entry applications for disabled people (from the keyboards described in section 3, it is for example WiVik and Fitaly One-Finger Keyboard). Disabled people are willing to learn the abbreviations, because it helps them to increase their typing speed significantly. Therefore, it would be appropriate to include such feature in the Keyboard application as well.

5.8.3 Keyboard Designer

The current implementation of the Keyboard application offers a very flexible way how to define the keyboard layouts. The user can define the position and content of every single key. The layout definitions are stored as XML files. However, if the user wants to modify an existing layout, or add a new one, he or she has to directly modify the content of the corresponding XML file. There is no graphical user interface for creating and modifying the keyboard layouts.

A WYSIWYG[4] editor for designing keyboards should be created, so that also users who do not know how to work with XML files could create and customize their own keyboards. The editor should allow the user to move and arrange the keys on screen to form the keyboard, and create the XML file for the user.

[4]WYSIWYG = What You See Is What You Get. This term means that what the user sees in the editor (shapes, text, colours, etc.) looks the same (or very similar) as what is then displayed in the application.

Chapter 6

Evaluation and Testing

This chapter evaluates the Keyboard application that was developed in this thesis, and whose implementation was described in the previous chapter. The five implemented keyboard types are compared to each other, as well as to the existing text entry systems (which were introduced in chapter 3). Results from usability testing with disabled people from the Jedlička Institute in Prague are also presented.

6.1 Comparison of the Developed Keyboard Types

Five keyboard designs were developed and are part of the Keyboard application. The designs were introduced in chapter 4.3; the realization of them was presented in chapter 5. The five implemented designs include the "Grid Keyboard", the "Keyboard with Coordinates", the "Shifting Keyboard", the "3x3 Keyboard", and the "Move-Controlled Keyboard".

This sections contains quantitative evaluation of the five keyboards. For the keyboards that can be controlled with a discrete input device (switches, joystick, etc.), it is possible to calculate the minimum number of button presses (or joystick moves) that are needed to write a piece of text. If the text was given to a real user, the number of button presses would most likely be higher, because the user would typically make (and then correct) typing errors. The numbers presented in this section therefore represent ideal values.

The number of button presses can also depend on the text itself. Two pieces of text of the same length, but with different content, can result in different total number of necessary button presses. For example, when typing with the "Grid Keyboard" (see section 5.5.1), it is necessary to navigate from one letter to another, and therefore it is the number of keys between each two subsequent letters what influences the total number of key presses. However, this is not true for all of the keyboards. In the "3x3 Keyboard", it is possible to select each key with exactly two button presses (or joystick moves), thus the order of the letters will not influence the total number of button presses.

It is also important how many "input actions" are used (available input actions were listed in section 5.4). For example, the navigation on the "Grid Keyboard" will be faster if all directions are used (i.e., if it is possible to navigate to any direction from each key), than if only two actions (typically, *Next* and *Previous*) are utilized.

The keyboards controlled with a continuous input device (e.g., a mouse) must be evaluated differently. In their case, just the number of "clicks" is not a sufficient measure of how fast typing with the keyboard will be. The distance the user has to travel with the pointer is also important. As we know from the Fitts' Law, the size of the target also matters. Keys that are farther can be easier (faster) to reach if they are at the edge of the screen, or are bigger than others. In this section, however, the evaluation will be for simplicity based solely on the number of clicks and the distance of keys, considering greater distance always as a negative factor.

6.1.1 Text for Evaluation

Since the Keyboard application is targeted to Czech users, a Czech text will be used for the evaluation. The text was chosen with respect to frequency of letters and words in (written) Czech language [6]. The text has 344 characters and was constructed to contain the most common Czech words. It is worded as follows:

Včera jsem na zastávce potkal svou dobrou kamarádku. Dlouhou dobu jsme se neviděli, celý poslední rok totiž pracovala v jiné zemi. Teď získala práci u jedné české firmy, která sídlí na Malé Straně. Říkala, že i když jsou v Praze vysoké ceny, chce se sem už v létě přestěhovat. Dala mi své nové telefonní číslo a domluvili jsme si další schůzku.[1]

6.1.2 Keyboards Controlled By Discrete Input

The Keyboard application contains four keyboards that can be controlled with a discrete input device: the "Grid Keyboard", the "Keyboard with Coordinates", the "Shifting Keyboard", and the "3x3 Keyboard". The default layouts included in the application will be used (pictures of them can be found in section 5). Layouts with alphabetical order of keys will be used in all keyboards.

Figure 6.1: Keyboard Application: Grid keyboard, letters in alphabetical order

[1] A translation to English is as follows: "I met my good friend at the bus stop yesterday. We have not seen each other for a long time, as she has worked in another country in the past year. Now she got a job at a Czech company that is located in Lesser Town. She said that even though the cost of living is high in Prague, she wants to move in there next summer. She gave me her new phone number, and we have arranged another meeting."

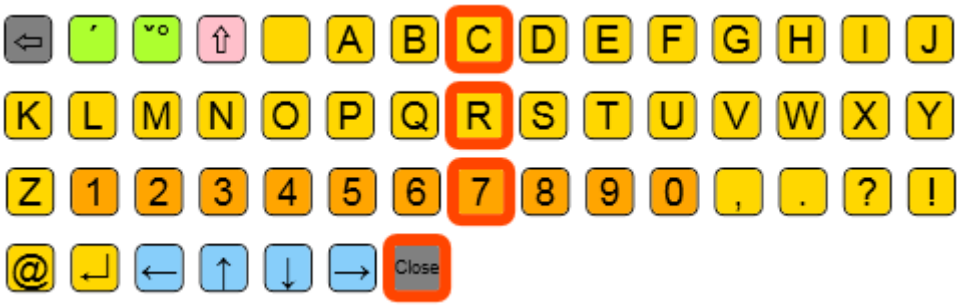


Figure 6.2: Keyboard Application: All keys from the default layout of the Shifting Keyboard (each row representing one keyboard "window" that the user can see)

keyboard type	**layout**	**used actions**	**# of presses**
Grid Keyboard	6.1	left, right, up, down + confirm	2336
		next row, next column + confirm	3434
		next, back + confirm	5880
Keyboard with Coordinates	5.5	next row, next column + confirm	2485
		8 directions (numbers)	797
Shifting Keyboard	6.2	next, back + confirm	4687
3x3 Keyboard	5.7	8 directions	780

Table 6.1: Number of button presses needed to write the test text using a discrete input device

Table 6.1 shows the number of button presses (or other equivalent actions, e.g. joystick moves) that need to be preformed to write the test text from section 6.1.1. Some keyboards, for example the "Grid Keyboard", can be used with different sets of actions. The table therefore contains multiple entries for these keyboards.

It was already mentioned above that the number of button presses depends also on the order of the keys (i.e., on the keyboard's layout). Even though all evaluated layouts had alphabetical order of keys, some differences between the keyboards could not be eliminated. For example, if the "Grid Keyboard" is configured to use the *Next*, *Back*, and *Confirm* actions, it will in fact become very similar to the "Shifting Keyboard", controlled with the very same actions. However, the number of button presses in table 6.1 is slightly different. That is because the layout of the "Shifting Keyboard" does not look the same as if the rows of the layout from the "Grid Keyboard" were put one after another in one line. Since the arrangement of the keys in the keyboard layout is important for the resulting number of button presses, table 6.1 also contains references to figures showing the keyboard layouts used for evaluation.

The data from table 6.1 show that the more outputs ("buttons") the device has, the less actions ("button presses") the user needs to perform to write the same piece of text. This is true in an ideal case, assuming the user is always able to choose the "shortest" way how to write a letter. However, the shortest way is not necessarily the most obvious one, which

means real users will possibly decide for a "longer" way in some situations, and thus the number of button presses will increase.

Let us give an example of such situation in the "Grid Keyboard" (see figure 6.1), with a four-directional navigation (i.e., actions *left/west*, *right/east*, *up/north*, and *down/south* are available). Since it is possible to "jump" from the last key in a row to the first one, the shortest way from the space key to the letter "S" is six keys long — "S" can be reached by pressing the *right/east* button six times. However, this path requires the user to go "away" from the target at first. That might not occur to the user, especially if he or she controls the keyboard for the first time. It can therefore happen that the user rather chooses the opposite way, i.e. going from the space key to the left. To reach the key with the letter "S" this way, the user needs seven button presses, which means the path is one key longer. Similar examples of such situations could be found in the "Shifting Keyboard" as well.

It is important to note that using longer paths does not necessarily mean the typing will be slower. The user can be faster if he or she does not think about the shortest path for too long, and rather simply chooses the path that is obvious to him or her at the moment. The speed will in general be individual for each user and will also depend on the user's level of experience with the keyboard. Some measurements of typing speeds of real users can be found in section 6.3.

Table 6.1 also shows that the number of button presses is approximately the same when using the same set of actions with different keyboard types. The more important will therefore be to choose the right device for each user, since different devices can produce the same actions (for example, a joystick and an eight-switch device are both able to produce eight navigation actions). Each user can also have different preferences of how the keyboard should look like — one can prefer seeing all keys at once, the other can be more comfortable with always having only one key on the screen.

6.1.3 Keyboards Controlled By Continuous Input

There are three keyboard types in the Keyboard application that can be controlled by a continuous input device — the "Grid Keyboard", the "3x3 Keyboard", and the "Move-Controlled Keyboard". Instead of navigating by moving a cursor, the user continuously moves the pointer on screen, and points to the key he or she wants the select.

Total distance the pointer has to travel can therefore be measured. All the evaluated keyboards were resized to have the same size on screen, so that the measured values could be compared the each other. The keyboards had 200 px in height and 800 px in width[2]. The number of "clicks" (or equivalent actions on other devices than mouse) can also be determined for the given piece of text. Table 6.2 shows both the path length and the number of clicks needed to write the test text on the three keyboard types.

Since the "Move-Controlled Keyboard" requires no clicks, it is the only one that is suitable for users that have problems performing such action. On the other hand, the path the pointer has to travel on this keyboard is the longest[3]. That is because the user always has to move

[2]Since the "3x3 keyboard" is squared, it was only 200 px wide.

[3]It would be hard to measure the shortest path the pointer has to travel on this keyboard exactly, because the user does not have to return the pointer to the middle of the confirmation area; it is enough to "touch" its edge to confirm the selected key. The path in this keyboard was therefore measured as if the user moved

keyboard type	layout	distance travelled [px]	# of clicks
Grid Keyboard	6.1	97013	390
3x3 Keyboard	5.7	59907	780
Move-Controlled Keyboard	5.8	136523	0

Table 6.2: Distance the pointer has to travel and number of clicks needed to write the test text using a continuous input device

the pointer "back" to the middle confirmation area, and also because the keys are placed only at the edges of the keyboard, so the empty space in the middle is quite large (which is in contrast with the "Grid Keyboard", where there is almost no empty space). The "3x3 Keyboard" has the shortest distance (since its layout is compact in comparison to the other keyboards), but requires the most clicks of all the keyboards (since two clicks are always needed to "press" a key).

The speed of typing with these three keyboards will depend on how fast the user is able to move the pointer and how long does it take him to make the pointer still and perform a "click". Depending on which of these times is the longest, the user should choose an appropriate keyboard. For example, the "3x3 Keyboard" will be suitable for users who are not able to move the pointer to long distances, or have problems with pointing to small targets, but will not be good for users that have problems performing clicks.

6.2 Comparison with Existing Text Entry Systems

It is not easy to compare the Keyboard application with other existing text entry system, either commercial or free. The main reason is that the Keyboard application is just a part of a larger application, EasyControl. That means the other text entry systems could not be probably easily used with the EasyControl, and the Keyboard application in turn is not able to communicate with other applications than those from EasyControl. However, during the development of the Keyboard application, implementation of a new feature for the EasyControl has started, which allows to send the output from the Keyboard application to external applications, such as MS Word, or an internet browser.

The main advantages of the Keyboard application is the configurability of the keyboard layouts. Assistants and teachers of the disabled users can create the layout according to their client's needs. It is possible to use pictures and sounds, change the text displayed on keys, choose different colours for the keys, and so on. Not many of the existing applications provide such flexibility, even though some of them (such as *Click-N-Type* or *WiViK*) offer layout designers or changing colours of the keyboard by using "skins". However, only few offer a way to create user-defined keys that can display any content (the *Clicker* application can do this, and it is in general probably the most configurable text entry system from all applications presented in chapter 3).

the pointer from each key straight back to the confirmation area, but just about 10 px past its border, and then to the centre of the subsequent key. In the other keyboards, the path was measured as total length of straight lines connecting the centres of subsequent keys.

Some important features that the existing text entry systems systems offer are also present in the Keyboard application, for example the automatic scanning and dwell clicking (both can be achieved by configuring the input from the EasyControl application). The Keyboard application also provides five different types of keyboards with different behaviour and way of controlling. None of the applications presented in chapter 3 have such feature.

A disadvantage of the Keyboard application is that there is no graphical interface for designing the layouts, i.e. they must be created by modifying or creating XML files "by hand" (in a text/XML editor). That might be a hard task for the people that work with the disabled users, since it requires a bit higher level of computer skills. However, implementing a graphical keyboard layout designer for the application is planned for the future, so hopefully there soon will be a tool that will help the users to create the layouts more easily.

A common disadvantage of the existing text entry systems that have all the necessary features for the disabled users is their price. The *Click-N-Type* keyboard is a configurable software keyboard, suitable for disabled users, that is free, but as the list of applications from chapter 3 proves, it is rather an exception. The rest of the applications suitable for disabled users is usually paid. The EasyControl suite, including the Keyboard application, is offered for free, which is a significant advantage for the users.

It can be seen as a disadvantage that the Keyboard application works mainly with the devices that were developed for the EasyControl application, which means that the support for an alternative device depends on whether it is supported by EasyControl. It might therefore happen that the application will not work with alternative input devices that require their own software to be able to communicate with the computer. An interface (either hardware or software) would have to be created and added for such devices. Apart from the special input devices developed by the NIT research group (see section 2.2), standard keyboard and mouse devices are also supported (including for example touchpads and trackballs).

The Keyboard application was developed for the .NET framework, which means it currently works only with Windows operating systems. This can be also seen as a disadvantage, since for example Linux and Mac OS users will not be able to use the application. However, since the whole EasyControl application was developed for this framework, it was necessary to choose the same technology for the Keyboard module as well. Providing support for other platforms than Windows would require big changes in the EasyControl application.

There are of course still many improvements that could be implemented in the Keyboard application. Because of the limited time for this thesis, it was unfortunately not possible to implement all of them, especially those that popped up in the later phases of development and testing. Despite of that, the Keyboard application contains most of the important features of both the commercial and free text entry systems, and in some ways is even more flexible than those systems.

6.3 Usability Testing

Since the main focus of this thesis is on the user interface of the developed application, it is natural that it should be tested with some real users. The usability testing was carried out in the Jedlička Institute for physically disabled young people in Prague.

Three sessions were arranged for the testing. An adult user and a child (first grader) took part in the first session, two third graders took part in the second session, and seven secondary school students participated in the third session.

All users were somehow able to use a physical keyboard in real life, even though some had problems with it (e.g. because their hands are shaking, or they are not able to use all fingers). Unfortunately, no users that are not able to use a physical keyboard were available for the testing, even though testing the Keyboard application with such users would be very valuable for the project.

Even though it is irrelevant for testing the Keyboard application whether the users use a wheelchair or not, it affected the testing at least in the way that it was necessary to adjust the height of the table with the PC and input devices for such persons. Some of the participants suffered from some kinds of concentration disorders, so it was rather the inability to complete the assigned task than the typing with the keyboard itself what was difficult for them.

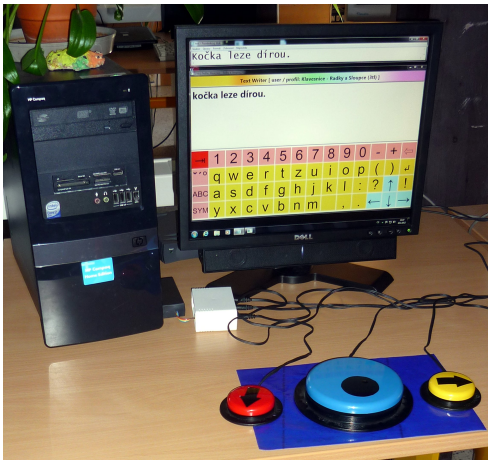
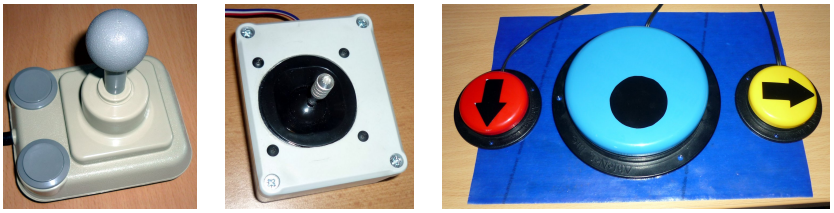


Figure 6.3: Set-up of the PC and switches for the usability testing

Figure 6.3 shows the setup of a table with a PC running the application and input devices for the testing. Two kinds of devices were used for the testing — a joystick and switches. Figure 6.4 shows all of them. There were some technical problems with the joystick shown in figure 6.4(a), so after one trial it was replaced with another type of joystick, shown in figure 6.4(b). Both joysticks were configured to produce eight actions (each for one direction). The other input device consisted of three switches and is shown in figure 6.4(c). The switches were configured to produce three actions — *Next Row* (the red switch), *Next Column* (the yellow switch), and *Action/Confirm* (the blue switch).

Since there were only three sessions for the testing (about three and a half hours of testing time), it was unfortunately not possible to test all five types of keyboards. Therefore only three were chosen — the "Grid Keyboard", the "Keyboard with Coordinates", and the "3x3 Keyboard". The "3x3 Keyboard" was controlled with the joystick, the "Grid Keyboard" and the "Keyboard with Coordinates" were controlled with the switches. The layouts for the testing were a bit different than the default ones — the most important difference is that they had no "Shift" key or the keys for diacritics. Capital letters and letters with diacritic



(a) Joystick 1 (b) Joystick 2 (c) Switches

Figure 6.4: Devices used for usability testing

were placed on a standalone layouts to which the user could switch from the default layout with lower case letters. The letters in the layouts of the "Grid Keyboard" and the "Keyboard with Coordinates" were arranged in the traditional *QWERTZ* order[4]. The test layouts can be found on the CD attached to this thesis.

For the purposes of the evaluation, each user was given an identifier, consisting of a letter and a number. The letter indicates the session the user took part in (*A* is for the first session, *B* for the second, and *C* for the third); the number simply indicates the user's order.

Before the testing itself, the users were given a short introduction to how the Keyboard application is controlled. They were also allowed to ask any questions during the testing (this was necessary, because some users tended to forget how to do things). Common task for all users was to write a given Czech sentence, "*Kočka leze dírou.*"[5], with at least two keyboard types. The users were not instructed that they have to correct all typing errors, so it was up to do user whether he or she will do so. However, not all of the users were able to complete the task. Some of the users managed to type the sentence with only one keyboard, and then they were too tired to continue. The youngest user (A2), could not read and write very well yet, so he was allowed just to type his name. The other user from the first testing session (A1) was given a different sentence. He managed to type it whole with the first keyboard type. He typed the first word only with the second keyboard type, and then he became too tired and did not want to continue, so the testing had to be ended.

Even though longer text would be better for the evaluation, since it would allow to better observe whether the users are improving in time, the chosen sentence has only three words. The main reason is that typing longer text would take too much time, which would lead to exhaustion of the users. The users would therefore probably not be able to test more than one keyboard type in one testing session, so there would have to be significantly more of these sessions.

6.3.1 Quantitative Results

During the testing, the Keyboard application was logging the times of all actions performed with the keyboard. These logs were then used to calculate various quantitative values. However, it must be noted that the quantitative results are not the most important outcome

[4] In Czech Republic, it is common to use the QWERTZ layout on both physical and software keyboards, instead of the original QWERTY layout.

[5] The sentence originates from a traditional Czech children's song. It means "a cat crawls through a hole".

user	keyboard type	avg. time to select one key	avg. # of actions to select one key
A1	coordinates	00:21.85	20.00
	grid	00:15.52	12.90
A2	coordinates	00:24.97	7.00
	grid	00:37.37	15.20

Table 6.3: Measured values from the usability testing, session A

of the testing. The more valuable results are the qualitative ones, obtained by observing the users, and interviewing them and their assistants (these results are discussed in the next section). Also, the fact that there were relatively few users that participated in the testing, and that their abilities varied, means that it is not possible to say, for example, which keyboard type is "the best", or to draw any similar conclusions. The needs of the users will always vary, which means that what is good for one user may be unsuitable for another. Since not all user have tested all keyboards, it is not possible to compare the keyboards based on averages calculated from all measured values. It is only possible to compare the keyboards based on results from the users that have tested the same keyboard types.

Tables 6.3 and 6.4 show measured values from the testing. The first session (A) has a separate table because in that session, the users were not given the same sentence to type, unlike the users in sessions B and C. If a row in a table is empty, it means that the corresponding user has not tested that particular keyboard type.

Both tables contain the average time and the average number of actions (switch presses or joystick moves) to select one key on the keyboard (i.e., to type a letter or other symbol, to switch the layout, or to press the *Delete* key). The "navigation mistakes" are also counted to the number of actions and the time to select a key (thus, the value of the average number of actions to select a key in the "3x3 Keyboard" can possibly be higher than two).

Table 6.4 also contains total time to write the sentence for each user and keyboard type[6], as well as an adjusted total time. The adjusted total time is the total time minus the time to correct typing errors the user has made. The adjusted total time therefore gives a better idea of how long would the user type the sentence if he had not made these errors. The reason for calculating this time is that the users who had not been correcting their errors would have got better (shorter) times, since they had not taken the time to correct their typing errors. The adjusted time should therefore eliminate the differences between the users caused by their attitude to correcting errors. Uncorrected typing errors are irrelevant to the adjusted time.

There are also averages of all the measured values in table 6.4. In sessions B and C, there were more users, and their abilities were more similar, so the average values are calculated. In session A, there were only two users, and they were very different, so it makes no sense to calculate averages for them.

Since the "Keyboard with Coordinates" and the "Grid Keyboard" were controlled by the same input device, and since layouts of both of the keyboards looked almost the same, the

[6]To save space in the table, the names of keyboard types are shortened — *coordinates* stands for the "Keyboard with Coordinates", *grid* stands for the "Grid Keyboard", and *3x3* stands for the "3x3 Keyboard".

user	keyboard type	total time	typing errors: not corrected	typing errors: corrected	typing errors: time to correct	adjusted total time	avg. time to select one key	avg. # of actions to select one key
B1	coordinates	05:56.11	0	0	00:00.00	05:56.11	00:15.48	10.96
	grid	07:29.01	0	0	00:00.00	07:29.01	00:21.43	9.50
	3x3							
B2	coordinates	08:51.06	0	3	01:19.14	07:31.92	00:18.97	15.18
	grid							
	3x3	06:51.69	1	1	00:07.67	07:29.01	00:21.67	2.89
C1	coordinates	05:16.36	0	1	00:17.54	04:58.82	00:12.49	11.32
	grid							
	3x3	10:43.94	0	17	03:01.95	07:41.99	00:12.15	2.32
C2	coordinates	15:28.20	1	14	07:42.36	07:45.84	00:17.04	13.43
	grid							
	3x3							
C3	coordinates							
	grid							
	3x3	07:23.68	1	6	01:42.17	05:41.51	00:13.38	2.34
C4	coordinates							
	grid	06:16.05	1	0	00:00.00	06:16.05	00:17.09	15.44
	3x3	13:22.84	13	8	01:26.88	11:55.96	00:17.84	2.41
C5	coordinates	05:26.32	0	5	00:39.80	04:46.52	00:09.89	10.58
	grid	04:29.40	0	4	00:16.47	04:12.93	00:08.34	12.55
	3x3	07:45.84	1	5	01:29.23	06:16.61	00:16.48	2.46
C6	coordinates							
	grid	02:56.22	0	0	00:00.00	02:56.22	00:07.66	9.70
	3x3	05:32.01	5	2	00:18.63	05:13.38	00:12.77	2.35
C7	coordinates	03:52.52	2	0	00:00.00	03:52.52	00:09.30	9.08
	grid	02:41.12	0	0	00:00.00	02:41.12	00:06.69	9.50
	3x3							
avg.	coordinates	07:28.43	0.5	3.83	01:39.81	05:48.62	00:13.86	11.76
	grid	04:46.36	0.2	0.80	00:03.29	04:43.07	00:12.24	11.34
	3x3	08:36.67	3.5	6.50	01:21.09	07:23.08	00:15.72	2.46

Table 6.4: Measured values from the usability testing, sessions B and C

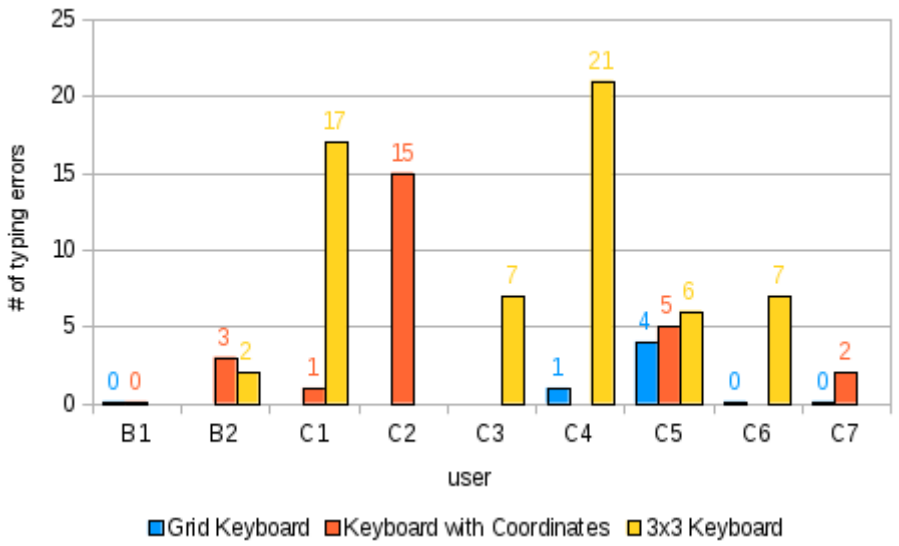


Figure 6.5: Graph showing typing errors made by users from testing sessions B and C

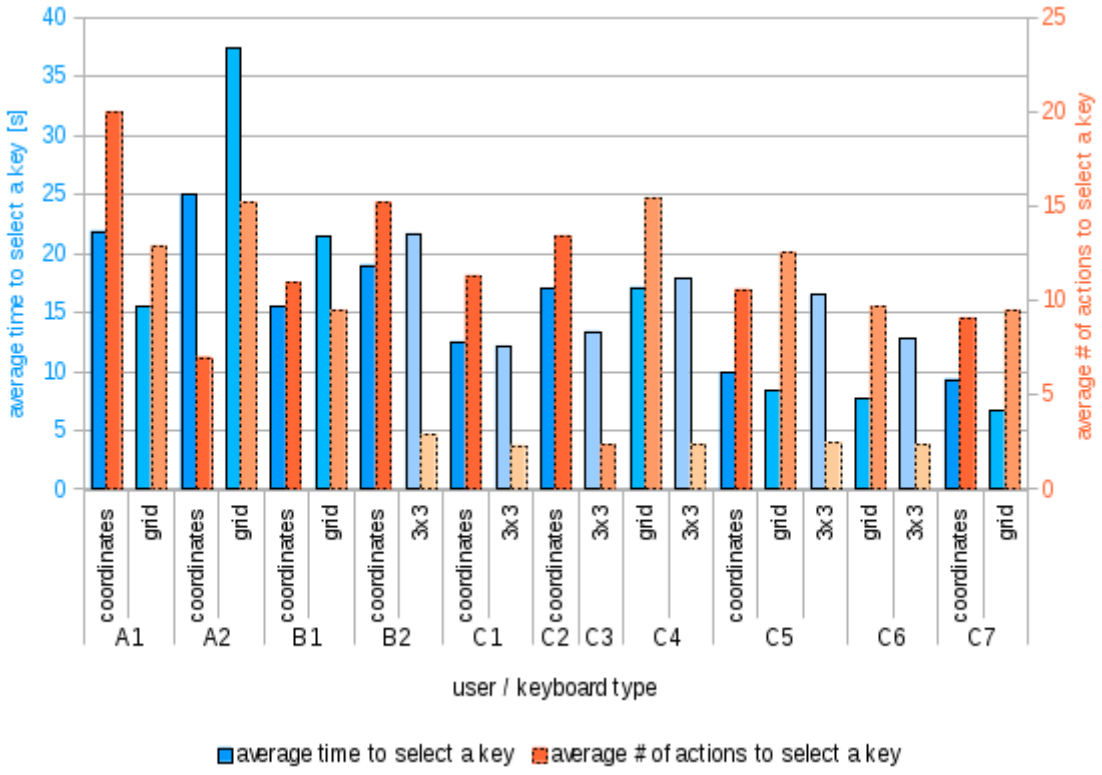


Figure 6.6: Graph showing average time and number of actions to select a key for all users

actions the users had to do to navigate from key to key were almost identical. The difference between the keyboards was that the whole row and column were highlighted when the user was navigating in the "Keyboard with Coordinates", while in the "Grid Keyboard", only one key was highlighted. The measured results from the testing show that the average number of actions to select a key are nearly the same for these two keyboards, which confirms that the navigation was done similarly.

Two users that have tested both of these keyboards had slightly shorter times with the "Grid Keyboard". A possible reason can be the learning curve, because those users that have tested more than one keyboard were always given the "Grid Keyboard" as the last one, which means they had been more familiar with the input device and/or the interface of the application by the time they started to type with the "Grid Keyboard". The user B1, on the other hand, had better time with the "Keyboard with Coordinates". That is because he had problems completing the task the second time — he got distracted a lot, could not focus on the task, and seemed tired.

In general, the measured values for the "Grid Keyboard" and the "Keyboard with Coordinates" are very similar. This is caused by the fact that, as mentioned above, the same input actions were used, and also by the fact that the keyboard layouts looked almost the same. Therefore, in this case, the user's preference of one of the keyboards over the other probably depends only on the way how the keyboard highlights the selected key (the "Grid Keyboard" was configured to change the colour and thickness of the border of the key; the "Keyboard with Coordinates" was set to change the colour of all keys in the corresponding row and column to light red, and change the colour of the key in the intersection to red). If different input actions had been used for each of the keyboards, there would probably be more differences in the keyboards. However, the configuration of the keyboards for testing had to be based on the available input devices.

Table 6.4 and corresponding graph in figure 6.5 show that the users have made more typing errors with the "3x3 Keyboard" than with the other two keyboards. The "3x3 Keyboard" was the one that was the least intuitive to control for the users, since it does not look any similar to a physical keyboard. The users also had to get used to manipulating the joystick, which took them some time. The results of testing the "3x3 Keyboard" are therefore also influenced by the input device itself. There were also some hardware-related problems with the joystick, like for example a slow reaction time. These problems negatively influenced the results, and will be further discussed in the next section.

Graph in figure 6.6 shows for all users the average time and number of actions they needed to select a key. Even though the average number of actions to select a key was lowest for the "3x3 Keyboard", most of the users had the worst total times, and in most cases also the average time to select one key, with this keyboard. Besides the possible hardware-related reasons, there are two other explanations of this. First one is that the layout of this keyboard displayed more characters than the layouts of the other two keyboards, and that the keys were not arranged in the traditional QWERTZ order. The "Grid Keyboard" and the "Keyboard with Coordinates" displayed 56 keys at once, while the "3x3 Keyboard" displayed 64 keys. Displaying more keys in an uncommon order caused that the users had to look for the desired key longer, thus increasing the total time. Even more significant delays were caused by making navigation mistakes. Since there were only eight actions (directions) the joystick could produce, there was no spare output to which a "cancel" action could be

assigned. Therefore, the automatic cancel had to be turned on — in case the user had selected a wrong tile, he or she had to wait for 10 seconds before the selection was cancelled and the first level of the keyboard appeared again. These 10 seconds of course significantly increased the total time to select a key. In the "Grid Keyboard" and the "Keyboard with Coordinates", no such waiting was necessary.

6.3.2 Qualitative Results

This section summarizes findings from the testing that have qualitative nature. These are results of a subjective observation of the users during the testing, done by the author of this thesis. Informal interviews with the users and their assistants or teachers were also a valuable source of feedback.

It is important to note that the results of the usability testing are applicable to the needs of the Jedličkа Institute in Prague. That is also the perspective from which the employees of the institute evaluated the Keyboard application when they were asked for an opinion. The institute focuses on young people and their education, so the Keyboard application could be used for educational purposes there. In case the Keyboard application is used for teaching children to read by using sounds in the keyboard, or helping them to recognize different letters by displaying the keys in different colours, the requirements for the application will of course differ a lot from the requirements of a user that would like to use the keyboard to chat with a friend, or to control a game. This has to be kept in mind when evaluating the users' (and other person's) comments that are presented in this section.

Feedback from the Users

The secondary school students that participated in the third session (C) of the testing were used to use a physical keyboard, so it is understandable that most of them indicated that typing with the Keyboard application takes too much time. They agreed that they would not want to use it for an ordinary work, for example to write their homework. However, some of them thought the keyboard could be useful for children that are not able to use a physical keyboard. They liked the "Keyboard with Coordinates" the best, and preferred the switches over the joystick.

The joystick was hard to use for most of the users, especially at the beginning. The problem was probably in the device itself. Before the joystick registered that the user had moved the lever, it was necessary to wait for about two seconds in the desired position, and only after that move the lever back to the default position[7]. Some users had problems to control the joystick this way, since the mechanism is not very intuitive. Their usual approach was to move the lever to the desired direction and then back immediately, which did not work. What is more, the keyboard software did not provide any visual feedback during the "waiting time", so the users did not know whether they actually hold the joystick in the

[7]The time delay has to be there due to the hardware implementation of the joystick, since it does not have separate outputs for the diagonal directions. Instead, these directions are defined as a sequence of selecting the two neighbouring straight directions. Therefore, to be able to distinguish a subsequent selection of two straight directions and a selection of a diagonal direction, the time delay was introduced. This time delay can be adjusted in the configuration of the input devices that EasyControl includes, but cannot be eliminated completely.

right direction. As a result of all this, and despite they have been given instructions before, some users did not understand how the joystick needs to be controlled. That indicates that the device was unsuitable for those users, and they would probably need a different kind of joystick, whose controlling would be easier and more intuitive. It would be also better if the keyboard could give some feedback to the user right after he moves the joystick (however, this was not possible to implement with the joystick device used for testing, since the waiting time was caused by the hardware — the Keyboard application did not receive any input action until the waiting time period expired).

Also, it was difficult for the users to select the diagonal directions, and it often happened that a "neighbouring" straight direction was selected instead, which increased the number of navigation mistakes. On the other hand, most of the users have improved during the testing, so about after typing the first word they have started to be faster with the joystick and made less mistakes. One user from the testing session *B*, a boy from the third grade, said that controlling the keyboard with the joystick is like a game, and he liked it better than the switches.

Even though the users eventually managed to work with the given joystick, the impression remains that a more suitable joystick type should be used together with the "3x3 Keyboard". The joystick device used for the testing was quite hard to control, but perhaps this disadvantage would be eliminated if the users had used the joystick for a longer time, and were better used to how it works. Actually, some of the users that participated in the testing were using a wheelchair controlled by a joystick. It would be good if a similar joystick could be used to control the keyboard, instead of the one that was used for the testing.

Only one user had problems with the principle of how the "3x3 Keyboard" works, i.e. that it is always necessary to move the joystick to the direction that corresponds to the position of a tile or a key in the keyboard. He tended to move the joystick to a wrong direction. For example, he moved the joystick towards himself (in south direction) even though he knew that the key he needs to select is in the north tile on the keyboard. He therefore needed some help with the "typing" — it was necessary to show him the direction where to move the joystick, and then he was able to repeat the movement with the lever of the joystick. After some time, he was able to type some keys without any help, but still some directions seemed to be "problematic" for him. This is an example of a user that would probably need a better visual indication of the joystick movement directly in the Keyboard application. Perhaps it would be better for him if the tiles were visually highlighted first, and selected only after some time (not immediately). The user would thus see whether he has highlighted the tile he wanted, and it would be possible for him to cancel the selection by moving the joystick back in case he sees that a wrong tile is highlighted.

The "Grid Keyboard" and the "Keyboard with Coordinates" were controlled by the users mostly without any problems. These keyboards are more intuitive for the users because they look more like a physical keyboard. They therefore seem to be especially suitable for one-time users. There were a few users that had little problems with the principle of highlighting and confirming a desired key in the "Keyboard with Coordinates". They selected either the correct row or column, but not both, and pressed the confirmation switch too early. Sometimes they also confused the switches and pressed a wrong one. Such problems are caused by the fact that the users have worked with Keyboard application for the first time, and would probably disappear if the users used the application more often.

When controlling the keyboards with switches, i.e. using the *Next Row* and *Next Column* actions, some of the users asked how they can get "back" in case they have missed the key they wanted to highlight. They were instructed that it is only possible to navigate "forward", i.e. that if they navigate through the entire row or column to its end, the cursor will then jump to the first key of the row or column again. The users found this annoying, and said they would prefer to have an option to go back directly. It is possible to configure the keyboard to do so, at least the "Grid Keyboard" type, but two more switches would be needed for that. However, only three switches were chosen for the testing. Moreover, not all users would appreciate having more switches. The adult user that tested the keyboard during session A had problems remembering which switch is for which action (even though the arrows on switches illustrated their function) and needed to be told this information over and over again. That was because he had a short-term memory disorder, so adding more switches would probably cause even more problems to him. That proves different users can have opposing needs, and the Keyboard application has to be prepared for adjusting to as much of them as possible.

Testing with the younger users was done with the sounds turned on (the application produced a sound for each typed letter or other character). The users reacted positively on that and liked the sounds. It seemed the sounds increased their interest in typing with the Keyboard application.

Feedback from the Teachers and Assistants

The teachers and assistants of the disabled users that took part of the testing also gave some opinions on the tested keyboards. They generally liked the "Keyboard with Coordinates" the best, since it gave the most noticeable visual feedback to the user. They also liked the idea of being able to use pictures instead of text on the keys, and the ability of the Keyboard application to play a sound for each key. They would also appreciate if the application could read whole words and sentences, which is unfortunately not possible with the current implementation.

The teacher of the third graders thought that the "3x3 Keyboard" is not very suitable for them, since it may be confusing that the user has to decide what tile he or she selects first, and then he actually types the letter. She also did not like that there are a lot of characters displayed in the layout of this keyboard — she was afraid that the user will have to look for the letter for too long, and that the layout will be confusing, since it does not look like a traditional physical keyboard. That is understandable, because she said she is teaching the children to get used to the layout of physical keyboards. She could therefore better utilize the "Grid Keyboard" or the "Keyboard with Coordinates" for that. She also noted that the users affected with spasticity would not be able to control the application with the joystick, since they could not hold the joystick still in the desired direction. Switches are more appropriate device for such users.

6.4 Adaptation to the User

Part of the assignment of this thesis also was to evaluate the ease of use of the Keyboard application for novice and expert users. In general, the traditional-looking "Grid Keyboard"

will be best for novice and one-time users, especially if they are familiar with a physical keyboard. The "Grid Keyboard" can be configured to have almost the same layout as a physical keyboard, and also the way of controlling it (by moving the cursor from one key to another) is quite intuitive, since it is used in other computer applications as well. The "Shifting Keyboard", controlled similarly, should also be quite easy to learn, even though the user will need some time to get familiar with the layout, since not all keys are displayed at once on the keyboard.

The "Keyboard with Coordinates" may require more time to master, depending on which way of controlling is used. Selecting the row and column by navigation is more intuitive, and requires the user to learn only three actions (*Next Row*, *Next Column*, and *Confirm*), but takes longer time. Entering the coordinates directly by their numbers is much faster, but requires the user to be able to learn how the eight numbers map to eight inputs of the hardware device he or she is using, which of course takes more time then learning how to use only three actions. However, if the user is able to control an input device that provides at least eight outputs, and takes time to learn how to use it, the way of entering the coordinates directly will be much faster.

The "3x3 Keyboard" and the "Move-Controlled Keyboard" will probably take the longest time to master. The principle how they are controlled is not that common, so the user will have to learn it to be able to use these keyboards efficiently. As the usability testing proved on the "3x3 Keyboard", it takes some time before the user fully understands how the keyboard works. That means that novice users will probably be slower with these keyboard than they would be for example with the "Grid Keyboard". However, this should be just a temporary phase. Once the user masters the controlling of the keyboard, his or her typing speed should significantly improve (these keyboards are therefore also suitable for users that need to write longer texts). More (and long-term) testing would of course be needed to prove this assertion.

To sum it up, it can be stated that the keyboards that are "slower" (e.g., require more actions to type one letter) are easier to learn than the keyboards with unusual way controlling. On the other hand, greater typing speed can be achieved with the latter group of keyboards.

The important factor that will influence how easy and fast is to use a particular keyboard type is also the input device that is used together with the application, which is closely related to the set of input actions that the keyboard will receive from the device. It is clear that typing with a keyboard controlled by a three-switch device is (theoretically) slower than if five-switch device was used. Not all keyboard types that the Keyboard application offers can be used with all kinds of devices. Some devices are also more suitable for a particular keyboard type than the others. For example, it will be probably better to use the "3x3 Keyboard" with a joystick than with eight standalone switches.

Last but not least, the configuration of the keyboard's layout plays very important role. Using a flexible definition of layouts, the application allows keys with basically any content to be placed on the keyboard. The person designing the layout can be for example a disabled user's assistant, who wants to build the layout to fit her client's individual needs, or a teacher that would like to use the keyboard in class for educational purposes. It is possible to create very small layouts, containing just a few keys, as well as large layouts, similar to those on physical keyboards. The keys do not even have to display text — pictures or symbols can be used instead. For example, it is be possible to create a layout containing keys displaying

pictures that stand for word categories (like “food”, “school”, “animals”, and so on). If the user selects one of the keys, a new layout appears where he or she can see the words from the selected category (either as text or pictures again), and can type them by selecting the corresponding key.

From what has been said so far, it is obvious that the Keyboard application allows to create basically any keyboard layout, and by choosing an appropriate keyboard type and input device, it should be possible to adapt the application to any user.

Chapter 7

Conclusion

A highly configurable text entry application, targeted to disabled users, was developed in this thesis. The application has a form of a software module that will be a part of the EasyControl software suite, developed by Nature Inspired Technologies Group at the Faculty of Electrical Engineering of Czech Technical University in Prague. It provides five different types of "keyboards", each with a different graphical user interface and behaviour. The keyboards were designed so that they suited both the standard and alternative input devices that the users may use together with the EasyControl application. The application was designed directly for EasyControl and its hardware devices, it is not meant to serve as a general-purpose on-screen keyboard.

Before designing these keyboard types, and also the four more that were not implemented, a survey of existing text systems was done, whose results are also presented in this thesis — some interesting representatives of both on-screen keyboards and alternative text entry systems were listed in chapter 3. The design of the keyboards is based on recommendations and guidelines published in research papers and articles from the domains of usability and user interface design.

The main advantage of the developed text entry application is the flexible configuration of layouts. It is possible to create custom layouts for each of the five keyboard types. The keys in the layouts can be configured to display any content — not only text, but also custom pictures can be used. The displayed content of a key does not even have to be the same as the output of that key. The number of keys that will be displayed on the keyboard can also be configured for four of the five keyboard types.

It is also possible to configure properties such as the background colour and the colour of text for each individual key. Besides that, there is a global configuration of colours that can be used to configure colours for specific group of keys (such as letters, digits, navigation keys, and others). There are other settings in the global configuration as well, including colours and thicknesses of keys' borders, the font the keyboard uses to display text, and others (complete list of settings can be found in section 5.7.3). It is therefore possible to create high-contrast layouts for users with poor vision as well colourful layouts for children.

It is simply up to the intentions and fantasy of the person creating the layout how the result will look like. Of course, there are things that cannot be changed (for example the shape of keys), but the offered amount of configuration options should be sufficient to adapt the keyboard and its layout to each individual user.

Most of the text entry systems available today, even those targeted to disabled users, only contain a traditional on-screen keyboard that looks like a physical keyboard (with additional features like auto scanning or dwell clicking). Typing with such systems is usually very slow. However, some novelty text entry systems exist, that provide an alternative (and faster) way of entering text. The application developed in this thesis also provides nontraditional ways of typing. The users can choose which one of the five available "keyboards" will suit them and their input devices the best.

Some usability testing was done with the Keyboard application, which took place in the Jedlička Institute for physically disabled young people in Prague. It was good to have a chance to try the application in a real environment. However, due to time constraints, only three of the five implemented keyboard types could be tested. The testing has shown that the visual feedback from the application is very important, and even though the application already contains it, it could be improved, especially for the keyboard type called "3x3 Keyboard". The testing has also proved that the choice of the input device that will be used for controlling the application is very important, and can significantly influence not only the user's typing speed, but also his satisfaction from using the text entry system.

As for the future work, there are three features that could be implemented for the Keyboard application. The first is an editor for the keyboard layouts with a graphical user interface. Currently, it is necessary to edit the layouts directly in their XML definition files, which is not ideal, even for people with enough computer skills. It would be better if the layouts could be designed in a graphical interface, so that the user immediately saw how the result will look like. The danger of making syntax errors when creating or editing the XML files would also thus be eliminated. The second feature is a prediction system, since it could significantly improve the typing speed with any of the keyboards. Suggestions of how the prediction could be incorporated in the design keyboards were presented in chapter 4, so at least some work in this area has already been done. The third proposed feature, that could also help the user to type faster, is an abbreviation expansion. This feature would allow the user to define abbreviations for frequently used words and phrases. Instead of typing every single letter of these phrases, the user could just write the corresponding abbreviation, and the application would "expand" it to the full phrase automatically.

A lot of interesting ideas for improvements and new features were also suggested by the people from the Jedlička Institute during the usability testing. Especially some kind of text-to-speech feature seems to be most needed by the disabled people nowadays, especially by those that have speech disorders, or cannot speak at all. It was also suggested that the application could be modified to be able to run on smart phones and tablets. The users could then carry the mobile device running the application with them, and it could help them to communicate with people around.

The world of disabled users is full of such requests, and all of them represent ideas for interesting projects. Hopefully more of these projects will be realized and will help the handicapped users to more easily access the computers and be able to better communicate with other people around them. This thesis aims to be one of these projects.

Bibliography

[1] ThickButtons website. http://www.thickbuttons.com, 2009.

[2] EdgeWrite project website. http://depts.washington.edu/ewrite/, 2010.

[3] 8pen website. http://www.8pen.com, March 2012.

[4] Fitaly website. http://www.fitaly.com, January 2012.

[5] OnScreen with Word Complete website. http://www.imgpresents.com/onscreen/onscreen.htm, March 2012.

[6] Statistical characteristics of the czech language. Published online at http://nlp.fi.muni.cz/cs/stat_cestiny (in Czech), March 2012.

[7] SwiftKey website. http://www.swiftkey.net, March 2012.

[8] Crick Software. Clicker website. http://www.cricksoft.com/uk/products/tools/clicker/home.aspx, March 2012.

[9] David MacKay. Dasher website. http://www.inference.phy.cam.ac.uk/dasher, February 2011.

[10] Exideas. MessagEase Onscreen Keyboard website. http://www.exideas.com/ME/ProductsMEOK.html, 2005.

[11] M. Fejtová, P. Novák, and O. Štěpánková. Easycontrol – universal control system. In *Computers Helping People with Special Needs*, volume 5105 of *Lecture Notes in Computer Science*, pages 1024–1029. Springer Berlin / Heidelberg, 2008. http://dx.doi.org/10.1007/978-3-540-70540-6_153.

[12] D. J. Gilman. AbilityHub: Assistive Technology Solutions. http://www.abilityhub.com/index.htm, April 2012.

[13] G. M. Hall. *Pro WPF and Silverlight MVVM: Effective Application Development with Model-View-ViewModel*. Apress, 1st edition, 2010.

[14] Holland Bloorview Kids Rehabilitation Hospital. WiViK website. http://www.wivik.com, 2010.

[15] Ken Perlin. Quikwriting website. http://mrl.nyu.edu/projects/quikwriting/, March 2012.

[16] Lake Software. Click-N-Type website. `http://www.lakefolks.org/cnt/`, December 2011.

[17] J. B. Lopes. Designing user interfaces for severely handicapped persons. In *Proceedings of the 2001 EC/NSF workshop on Universal accessibility of ubiquitous computing: providing for the elderly*, WUAUC'01, pages 100–106, New York, NY, USA, 2001. ACM.

[18] Nature Inspired Technology Group. Tools and applications for users with limited movement and reaction abilities, 2010. `http://nit.felk.cvut.cz/projects/pomucky`.

[19] S. B. Nesbat. A system for fast, full-text entry for small electronic devices. In *Fifth International Conference on Multimodal Interfaces*, Vancouver, November 2003. `http://exideas.com/ME/ICMI2003Paper.pdf`.

[20] J. Nielsen. Ten usability heuristics, 2005. Published online at `http://www.useit.com/papers/heuristic/heuristic_list.html`.

[21] OATS. DKey website. `http://www.oatsoft.org/Software/dkey`, March 2012.

[22] Origin Instruments. Keystrokes 4: On-Screen Keyboard for Mac OS X. `http://www.orin.com/access/keystrokes/index.htm`, March 2012.

[23] P. Novák et al., Nature Inspired Technology Group. Jak umíme pomoci lidem s omezenou schopností pohybu a reakce?, November 2010. Not published.

[24] K. Perlin. Quikwriting: Continuous stylus-based text entry. In *11th annual symposium on User Interface Software and Technology*, San Francisco, CA, November 1998. `http://mrl.nyu.edu/perlin/doc/quikwriting/quikwriting.pdf`.

[25] Sensory Software. Grid Keys website. `http://www.sensorysoftware.com/gridkeys.html`, March 2012.

[26] Sensory Software. The Grid 2. `http://www.sensorysoftware.com/thegrid2.html`, March 2012.

[27] B. Shneiderman. Universal usability. *Communications of the ACM*, 43(5):84–91, 2000.

[28] B. Shneiderman and C. Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Pearson Addison Wesley, 4th edition, 2004.

[29] J. Smith. *Advanced MVVM*. Josh Smith, 2010.

[30] B. Tognazzini. First principles of interaction design. Published online at `http://www.asktog.com/basics/firstPrinciples.html`.

[31] O. Vavroušek. TapTap Keyboard website. `http://www.taptapkeyb.ic.cz`, 2007.

[32] M. Vogal. Možnosti predikce textu při psaní osobou s omezenou pohyblivostí a reakcí. Master's thesis, Czech Technical University in Prague, May 2011.

[33] J. O. Wobbrock, S. K. Kane, K. Z. Gajos, S. Harada, and J. Froehlich. Ability-based design: Concept, principles and examples. *ACM Transactions on Accessible Computing*, 3(3):9:1–9:27, April 2011.

[34] J. O. Wobbrock, B. A. Myers, and J. A. Kembel. Edgewrite: A stylus-based text entry method designed for high accuracy and stability of motion. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '03)*, pages 61–70. ACM Press, November 2-5 2003.

Appendix A

User Manual

A.1 Running the Application

The keyboard is not a standalone application, it is just a part of the EasyControl program suite. It is included in some of the applications EasyControl provides. It is therefore always necessary to run one of the applications from EasyControl first to see the keyboard.

A.2 Controlling the Keyboards

There is a choice of five different keyboard types. Each of them is controlled differently. These five keyboard types have been described in sections 4.3 and 5.5 of this thesis. Please refer to these sections for explanation of how each of the five keyboard types works and what are the possibilities to control them. The description is written in a form that should be acceptable for everyone, so it would be redundant to include the explanation here again. However, there is one thing that the user should be aware of before reading that section, and that is the mapping of outputs from the hardware devices to actions in the keyboard. This mechanism is explained below, in section A.2.1.

A.2.1 Explanation of Input Actions

To be able to control any of the keyboard types properly, it is necessary to be aware of how outputs from the hardware devices (such as switches, joystick, and others) are mapped to actions in the keyboard.

Each device typically provides one or more "outputs". Let us give some examples. A single switch provides one output, which is trigged by pressing the switch. A device consisting of three switches provides three distinct outputs, each from one of the switches. A joystick provides eight outputs, each corresponding to one direction to which the lever of the joystick can be deflected. In case the joystick has some additional buttons, the number of its outputs can of course higher.

But how do EasyControl know how to interpret these outputs? That has to be configured of course! The EasyControl application has some predefined actions, that are used to tell the keyboard what to do. These actions can be assigned to the outputs of the connected

input device by means of the configuration software. It is therefore possible to define that, for example, deflecting the lever of the joystick to the right will mean "jump to the next key in the keyboard".

Not all actions have to be utilized by the currently used input device. In fact, it is more usual that only some of them are mapped to the outputs of the device.

There are currently the following fifteen actions provided by EasyControl:

- *Right*, *Left*, *Down*, and *Up*. These are actions typically used for navigation to the four basic directions. They can tell the keyboard to move the cursor (= focus) in the corresponding direction.
- *Right-Up*, *Left-Up*, *Right-Down*, and *Left-Down*. These are also navigation actions. They can be used for navigating to the diagonal directions.
- *Next Item* and *Back Item*. These are actions for "simple navigation". They only tell whether the direction of the navigation should be "forward" (*Next Item*), or "backward" (*Back Item*).
- *Next Row* and *Next Column*. These actions can also serve for simple navigation. They tell the application that the cursor (focus) should be moved to the next row or column, respectively.
- *Action*. This action is usually used for confirmation of the currently focused item (e.g., a key) in the application (its functionality is similar to the *Enter* key on a physical keyboard).
- *Escape*. This action is typically interpreted as "cancel" or "exit".
- *None*. This is an empty action and does nothing. It is ignored by the keyboard.

A.3 Configuration

The Keyboard application can be configured in a global configuration dialogue that can be opened by pressing the *F1* key (on a physical keyboard).[1] Figure A.1 shows a screenshot of the configuration dialogue, with the tab with configuration of the keyboard selected. The dialogue in the screenshot contains Czech text, since the EasyControl application is currently not localized in any other language. In this manual, however, English translations of the texts will be provided when referring to the settings.

Note that different names are used for the keyboard types in the configuration dialogue than in the text of this thesis. The "Move-Controlled Keyboard" is referred to as *CircuitKeyboard*, the "3x3 Keyboard" as *MatrixKeyboard*, and the "Keyboard with Coordinates" as *AxisKeyboard*. The "Shifting Keyboard" and "Grid Keyboard" are the same — *ShiftingKeyboard* and *GridKeyboard*, respectively.

A.3.1 Selecting Keyboard Type

The keyboards can be switched by using the combo box menu at the top of the configuration dialogue. Simply select the desired keyboard and press the *OK* button.

[1] It is possible that this will change in future versions of EasyControl, so please refer to the EasyControl's user manual or help for up-to-date information.

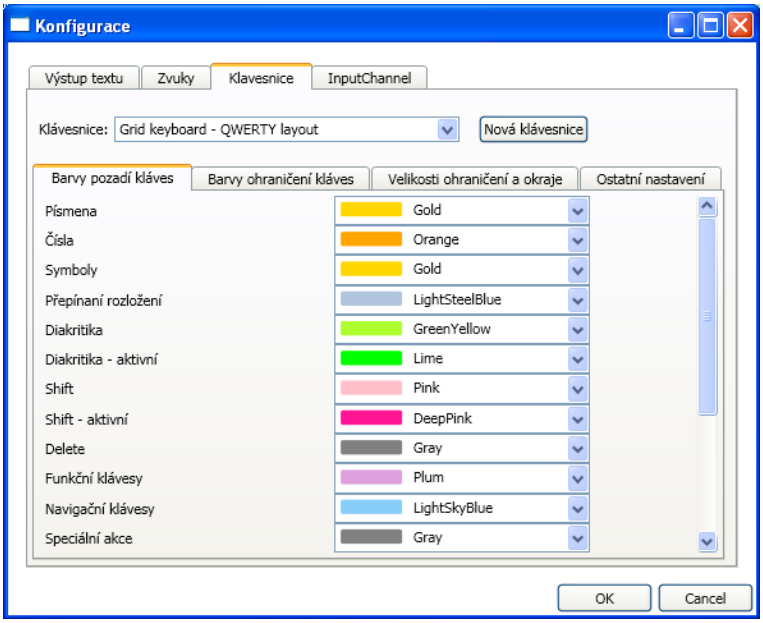


Figure A.1: Global configuration dialogue that contains configuration of the keyboard

The items that are be displayed in the list of available keyboards are defined in the configuration file called `Keyboards.xml`. How a keyboard can be added to (or removed from) the list of available keyboards is explained in section A.3.6.

A.3.2 Changing Visual Appearance

The configuration dialogue contains a lot of settings that can be used to adjust the keyboard's visual appearance. Any changes will take place when the *OK* button is pressed, or can be cancelled by pressing *Cancel*.

The settings are grouped into four tabs:

- Background colours (*"Barvy pozadí kláves"*). This tabs contains "global" settings of background colours of keys. Background colour can be changed for each of the group of keys (such as letters, digits, navigation keys, special actions, etc.) as well as for some special controls used in the application (e.g., the middle confirmation area in the "Move-Controlled Keyboard").

- Colours of borders (*"Barvy ohraničení kláves"*). This tab contains settings of border colours.

- Thickness of borders and margins (*"Velikosti ohraničení a okraje"*). Thickness of borders of keys and other controls, as well as the space between keys, can be configured here.

- Other settings (*"Ostatní nastavení"*). This tab contains various other settings, like for example keyboard's font, automatic cancel timeout, turning sounds on/off, and others.

Some settings are specific just to a particular keyboard type. In that case, the type of the keyboard is stated in parentheses before the name of the option.

A.3.3 Creating New Layout

Before you can start to design a new layout, a new file with its definition has to be created. That can be done either manually, by copying and renaming an existing keyboard definition file, or from the configuration dialogue.

If you want to do it the second way, press the button named *"Nová klávesnice"* (New keyboard). A small dialogue will appear that allows you to specify the name of the keyboard and its type, and also the name of the file with the definition of the keyboard's layout. After filling in these three input field, press the *OK* button. A new file with the specified name will be created in the configuration directory. The file will contain a template for the selected keyboard type, with all keys empty. You can then edit the file as you like in your favourite XML editor (see section A.3.4 for explanation of how to design a layout).

When using the configuration dialogue to create a new keyboard, the keyboard will be automatically added to the list of available keyboards. If you create a new keyboard manually, you will have to do this yourself (see section A.3.6 for explanation of how to do that).

A.3.4 Designing Layouts

The XML file containing the definition of the keyboard (or, better said, its layout), has the following structure for all five keyboard types:

```
<?xml version="1.0" encoding="utf-8" ?>
<Layout Name="...">
   ... here is a place for keyboard type specific definitions ...
    <Keys>
    ... individual keys are defined here ...
    </Keys>
</Layouts>
```

Keyboard Type Specific Definitions

Some keyboard types require that the `Layout` element in the definition file contains an element called `SizeDefinition`. The nested elements in this element define configurable parameters of the keyboard.

Size definitions for *GridKeyboard* and *AxisKeyboard*:

```
<SizeDefinition>
    <Rows>number of rows in the keyboard</Rows>
    <Columns>number of columns in the keyboard</Columns>
</SizeDefinition>
```

Size definitions for *ShiftingKeyboard*:

```
<SizeDefinition>
    <Columns>number of visible keys</Columns>
</SizeDefinition>
```

Size definitions for *CircuitKeyboard*:

```
<SizeDefinition>
    <Vertical>number of keys to the left/right of the confirmation
    area</Vertical>
    <Horizontal>number of keys above/below the confirmation
    area</Horizontal>
</SizeDefinition>
```

Definition of a Key

The element `Key` is used to define a key. Any number of these elements can be placed inside the `Keys` element. However, if you specify more keys than is the defined size of the layout, the extra keys will be ignored.

The definition of a key has the following structure:

```
<Key>
    <DisplayString>...</DisplayString>
    <DisplayImage>...</DisplayImage>
    <Output>...</Output>
    <ShiftOutput>...</ShiftOutput>
    <Sound>...</Sound>
    <BackgroundColor>...</BackgroundColor>
    <TextColor>...</TextColor>
    <OutputColor>...</OutputColor>
    <Type>...</Type>
</Key>
```

As you can see, there can be nine elements inside a `Key` element. Some of them are optional, some are not. Their meaning is as follows:

- `DisplayString`
  - its content defines the text that will be displayed on the key
  - the text can be of any length (or empty), and consist of any characters (it is also possible to use XML entities, like for example "@" for the at sign)
  - mandatory element
- `DisplayImage`
  - its content defines the path to an image that should be displayed on the key (use just the name of the file in case the image is placed in the same directory as the definition file)

  - *BMP*, *GIF*, *JPEG*, *PNG*, and *TIFF* formats are supported
  - if the image is set, it will always be used instead of the text, unless displaying images is disabled in the configuration
  - it is recommended not to leave the `DisplayString` element empty even if the `DisplayImage` element is used (the reason is that if the file with the image is not found or cannot be read, the text can be used instead)
  - optional element

- `Output`
  - its content defines the "output" of the key (empty content is allowed)
  - what the value of the output means depends on the type of the key (see the description of the `Type` element below) — for keys like letters and digits, the output is the text that should be typed if the key is pressed; for "action" keys, the output is the name of the action; for keys that switch layouts, the output is the name of the file defining the layout; etc.
  - mandatory element

- `ShiftOutput`
  - its content defines the output of the key if the "Shift" mode is on
  - optional element, ignored for action keys

- `Sound`
  - its content defines a path to a file with a sound that should be played when the key is pressed
  - only *WAV* format is supported
  - optional element

- `BackgroundColor`
  - its content defines the background colour of the key as a hexadecimal RGB[2] value, for example "FF0000" for red
  - optional element (if the colour is not specified, global settings will be used)

- `TextColor`
  - its content (a hexadecimal RGB value) defines the colour of the text that is displayed on the key
  - optional element (if the colour is not specified, global settings will be used)

- `OutputColor`
  - its content (a hexadecimal RGB value) defines the preferred colour of the key's output (i.e., of the typed text)
  - optional element

- `Type`
  - mandatory element whose content defines the type of the key
  - only the following values are allowed (for the explanation of their meaning, see section A.3.5): *Letter*, *Digit*, *Symbol*, *Diacritic*, *Shift*, *SwitchLayout*, *Delete*, *Control*, *Navigation*, *Action*, and *Extra*

[2]RGB stands for "Red, Green, Blue". It is a commonly used colour model.

A.3.5 Available Types of Keys

There are nine types of keys which can be placed in a keyboard's layout.

Letter, *Digit*, *Symbol*

These three types of keys are used simply for typing text. The value of their `Output` defines the character or text that will be typed if they are pressed.

For keys of type *Letter*, it is not necessary to define the `ShiftOutput` element. The output of these keys will be converted to upper case automatically if the "Shift" mode is on.

Shift

If a key of type *Shift* is pressed, it will turn the "Shift" mode on (or off, if it is pressed again). The "Shift" mode is typically used to type upper case letters, or whatever the other keys have in their `ShiftOutput` elements. The "Shift" mode is automatically turned off after the next key is pressed.

Diacritic

Keys of type *Diacritic* can be used in a similar way how the keys with diacritics are used on a physical keyboard. If such key is pressed, the keyboard "remembers" the diacritical mark, and then combines it with the output of the key that is pressed next, if that is possible. For example, if a key with the caron ("ˇ") is pressed, and then key with letter "r" is pressed, the result will be the letter "ř". If combining the diacritical mark with the next key is not possible (for example, if the next pressed key is "5"), the diacritical mark will simply be printed in front of the other character (the result of the previous example would therefore be "ˇ5").

The value of the `Output` of these keys has to be the diacritical mark. It is also possible to include two diacritical marks in one key. The way how to do this is to put the other diacritic to the `ShiftOutput` element. The other diacritical mark could then be typed in the "Shift" mode, the first one in the standard mode.

Only diacritics used in Czech alphabet are supported by these keys, which means there are three possible diacritical marks that can be used — the acute (’), the caron (ˇ), and the ring (°).

The other possibility of typing letters with diacritic is to place them in the layout directly, i.e. by means of the keys of type *Letter*.

Delete

Keys of type *Delete* are used to delete the last typed character (in case the `Output` is set to "DeleteOne"). It is also possible to define a key that will delete everything that has been written (the "DeleteAll" action can be used for this).

SwitchLayout

Keys of type *SwitchLayout* are able to change the current layout of the keyboard. That means the type of the keyboard remains the same, but its definition (size parameters and collection of keys) will be loaded from another definition file. The name of this file (or a relative path to it) has to be specified inside the `Output` element in the definition of the key.

Navigation

These are keys used for navigation (moving the cursor). Their functionality is similar to the cursor keys on a physical keyboard. You can use "Left", "Right", "Up", or "Down" as the value of the `Output` element in these keys' definition.

Control

The *Control* key type can be used to define special keys. Currently, "Ctrl", "Alt", and "Tab" are supported by EasyControl, which means you can specify these values in the `Output` element of these keys.

Action

Keys of type *Action* can be used to place any predefined user action on the keyboard. These predefined user actions have to be supported by EasyControl, so please refer to its current user manual or help to get the list of them. In the version in which the Keyboard application was introduced, "Close", "Next", "Prev", and "Enter" actions were available.

Extra

Keys of type *Extra* define a placement of actions that are supplied by the application that uses the keyboard. Such actions can even change with the state of the application. The content of `DisplayString`, `Output`, and `ShiftOutput` elements is therefore ignored for these keys, since these values are supplied in runtime.

Let us give an example of these *Extra* keys in action. Consider an e-mail client application, that consists of a contact list, a list of recipients, and a window to write the message to. If the cursor is on the contact list, actions like "Add to recipients", or "Create new contact" will be probably needed. However, if the user moves the cursor to the window where the message is written, such actions do not make sense any more. Different ones are needed instead, such as "Send", or "Save to concepts". This is where the *Extra* keys come in handy. You can simply place the keys anywhere on the keyboard, and each time the user switches "focus" to another component and the application supplies the keyboard with a new set of actions that the user may need in the new context (if necessary), these actions will be displayed on the *Extra* keys.

A.3.6 Adding, Removing, and Renaming Available Keyboards

List of available keyboards (which will be displayed in the configuration dialogue where the keyboards are selected) is defined in the *Keyboards.xml* configuration file. This file can be found in the main configuration directory (which is by default called `Keyboards`, and is placed in the same directory as the application's executable).

If you want to add or remove a keyboard from the list of available keyboards, you can do so by adding or removing the corresponding entry in this configuration file.

The *Keyboards.xml* file has the following structure:

```
<?xml version="1.0" encoding="utf-8" ?>

<Keyboards>

    <Keyboard>
        <Name>...</Name>
        <Type>...</Type>
        <LayoutFile>...</LayoutFile>
    </Keyboard>

      ...

</Keyboards>
```

Each `Keyboard` element defines one keyboard. A keyboard has three properties, defined by the nested elements of the `Keyboard` element:

- `Name` — its content defines the name of the keyboard, that will be displayed in the list of keyboards in the configuration dialogue. It is therefore recommended that the name is unique and identifies the keyboard sufficiently (names like "Shifting keyboard with alphabetically ordered keys", "Black & White Keyboard", or "Peter's keyboard" are just some examples). If you just want to rename an existing keyboard, you can simply edit the content of this element in the corresponding keyboard entry.
- `Type` — its content defines the type of the keyboard. The value must correspond to one of the five types the application provides: *GridKeyboard*, *ShiftingKeyboard*, *AxisKeyboard* (for the "Keyboard with Coordinates"), *MatrixKeyboard* (for the "3x3 Keyboard"), or *CircuitKeyboard* (for the "Move-Controlled Keyboard").
- `LayoutFile` — its content defines the name of (or a relative path to) the file with the keyboard's definition. If keys that can switch layouts are used in the keyboard, there will be typically multiple files for one keyboard. However, it is enough to specify the "starting" layout in this element, the rest of the files need not be explicitly specified here.

Appendix B

Content of the Attached CD

.
- **bin**
  - TextWriter.zip - archive containing a sample application with the keyboard
  - howto.txt - information how to run and control the sample application
- **doc**
  - **html** - documentation of the source code
    - index.html - starting point for viewing the documentation
- **src**
  - **Keyboard** - source codes of the Keyboard application
  - readme.txt - important information about the source codes
- **testing**
  - **layouts** - keyboard layouts used for usability testing
  - **logs** - logs from the usability testing
- **text**
  - **latex** - LaTeX source files of this text
  - fiedlkla-dp.pdf - this text
- dp-info.txt - short information about this thesis (including the abstract)
- index.html - content of the CD with links and other information
- thesis-assignment.pdf - scanned version of the official assignment of this thesis