DEVELOPING GRAPHICAL USER INTERFACES USING HUMAN-COMPUTER INTERACTION PRINCIPLES

Olga Casian, Alexandru Railean
Technical University of Moldova
dae.eklen@gmail.com, ralienpp@gmail.com

Abstract. *With the development of modern programming environments, designers and programmers have a large palette of various widgets; however, having them is not enough for designing an efficient graphical user interface. A user centered design approach that follows the principles of human-computer interaction can improve usability and increase the speed of mastering any software. This article describes the designing process and motivations of the interface that was made for an e-learning plug-in for the Pidgin instant messenger.*

Keywords: *human-computer interaction, graphical user interface, usability, Qt, Python, UX.*

I. Introduction

As humanity uses computers more extensively, the software becomes more complex, and so do the interfaces. Old fashioned command line interfaces cannot always provide users enough possibilities to interact successfully with the software, nor can they adequately reflect the current system's state in a friendly manner. In contrast, modern graphical user interfaces (GUI) provide mechanisms to implement ideas that were impossible to accomplish before.

This article tells about the developing of the interface of an e-learning plug-in for Pidgin [1]. Among the plugin's key features are: a shared canvas and audio sessions, as well as the possibility to save the current session and resume it later. The goal was to create an accessible user-friendly interface that should help users master the software faster.

There are three types of the conceptual models:

- the user's mental model – what people *think* the program does
- the system image – what the system *really* does
- the designer's conceptual model – how the designer maps the system's state to the interface.

The goal of the designer should be to help people use the software successfully by producing the results they want with minimal effort and no emotional stress. The only way the designer communicates to the users is through the system image, so that means that the designer should have the same mental model as users [2]. The system image, the actual model, of the good interface has no inconsistencies with the user's mental model. This can be achieved by respecting human-computer interaction (HCI) principles.

II. Human-computer interaction concepts

There are a lot of principles that the designer should take into account for the best practices. Among them the most important ones are:

- ***Steep learning curve.*** The notion was first introduced in [3] and describes how fast one is able to learn the information, the overall amount learned versus total resource invested; applying it to interfaces – how fast one can master the program. For example, the Notepad program on the Windows operating system (OS) is very easy to learn due to its simplicity. In contrast, Microsoft Office Word is more difficult to learn, but it offers a larger variety of features available after user

masters the software. In this case Notepad has the curve with the large initial increase while Word has a shallow curve at the beginning because of its complexity. Ideally the curve should be steep, which demonstrates that the learning is achieved rapidly. Sometimes the product complexity can negatively affect the shape of the curve; in these cases the progressive disclosure principle can improve the situation.

- ***Progressive disclosure.*** This principle is frequently used in complex products that have a large feature set. According to it, only the most important features and options are showed first, to avoid information overload and make it easy for people to make a choice. However, the larger set of the specialized features is available upon the user request while in the majority cases the initial set of features is enough to accomplish the task [4]. Having the set of reasonable default values can also refer to this principle.
- ***Hick's law***. The time it takes to make a decision is a function of the number of options that are available. If people are shown less options, they can choose faster [5].
- ***Fitts' law.*** The law states that the time to acquire a target is a function of the distance to the target and the target's size. The farther you are and the smaller is the target, the longer it takes to navigate the cursor to it; and vice versa. Taking into account the law, the size of the control should be proportional to its expected frequency of use [5].
- ***GOMS*** (Goals, Operators, Methods, and Selection rules) is the quantitative method that is able to determine the complexity of the task and express the time it takes to complete it in an exact, numerical form. The keystroke-level model operates by assigning each elementary action (keying, pointing, homing, mentally preparing and responding) a certain value. GOMS is useful in cases when the designer has more than one solution to accomplish a certain task - the method points out the optimal solution, the one with minimal number that corresponds to the elementary actions [5].
- ***Consistency.*** According to this principle, the software should meet the user's expectations. There are several levels of consistency:
  - interpretation of the user's behavior (for standard operations program uses the same set of shortcuts as are set in the system),
  - consistency with itself (look and behavior of all elements should follow the rules that apply at least to the whole product),
  - platform consistency (following the human interface guidelines (HIG) of the target OS to improve the user experience),
  - cross-platform consistency – such that the software looks similar in different environments (Windows, Linux with KDE, Linux with Gnome, etc).
- ***Forgiving*** programs and interfaces are those which protect the user's work ensuring that the users will never lose their work or data due to losing Internet connection, having sudden loss of power or running out of space. This principle mostly refers to the program architecture; however the interface is able to enforce a certain kind of behavior. A good example is the undo functionality – which enables people to repair a mistake; or the automatic saving of unsaved data when the program is closed, without asking a confirmation. Basically, any operations that can potentially lead to loss of data must be reversible.

III. Environment details

The interface was developed in the Qt framework [6] and the Python programming language, on Linux based OS LinuxMint. Qt is widely used cross-platform software for creating applications and consistent GUIs. Currently maintained by Nokia, Qt is a free and open source framework with APIs for C++. However there are bindings for other programming languages, including Python, a powerful interpreted language with a high level of abstraction that is ideal for rapid cross-platform application development.

IV. Implementation

Coding more complex layouts can become pretty complex, that is why the main part of the interface was done in Qt Designer, the Qt framework's tool for building GUIs. Initially, the interface was created according to the previously described principles in a what-you-see-is-what-you-get (WYSIWYG) manner.

In this case the interface contains images that should be added into the project as resources. It is possible to add icons in the code directly, but the right way to do it is to add resources at the stage of working in the Qt Designer. The resources will be converted later to the Python file that will be included as a usual Python module. For obtaining the module they should be compiled using the *pyrcc4* command line tool.

After the widgets were arranged and all the options were set, in order to make GUI looking native in any OS one should add layouts. Then one should set the proper tab order for those users who use only keyboard to navigate.

After all the previously described steps it is possible to launch the application from Python by the following minimal code:

```
import sys
from PyQt4 import QtCore, QtGui
import resource.res                                    # include resource file

class MainWindow(QtGui.QMainWindow):
        def __init__(self, parent=None):
                super(MainWindow, self).__init__(parent)    # calling parent's
                                                            # constructor
                uic.loadUi('project.ui', self)              # loading file made in
                                                            # Qt Designer

if __name__ == "__main__":
        app = QtGui.QApplication(sys.argv)
        myApp = MainWindow()
        myApp.show()                                        # show application
                                                            # window
        sys.exit(app.exec_())                               # exit application
```

The code later will contain other functionality that was impossible to add at the previous steps. Some elements, such as the help dialog, were completely created in the code.

V. Human-computer interaction principles compliance

The GUI follows previously mentioned concepts in the following way:

- The interface has a steep learning curve due to simplicity of the software and application of the progressive disclosure principle.
- Reasonable default settings ensure basic tasks can be accomplished swiftly. It is easy to change the settings, enabling power users to tweak the program to their needs.
- Following the GOMS principle, in order to perform main tasks, starting graphical and audio sessions, user just needs to start drawing and\or speaking to other session members avoiding any additional operations. To accomplish other potentially frequently used tasks, such as as changing the color or size of the brush, handy shortcuts were introduced.
- The interface is consistent with the usual user behavior as it uses the same shortcuts for common operations (ex.: F1 for Help, Ctrl+Z for undo the latest stroke, Ctrl+Q for quitting the program and others), with itself, having similar widgets for similar operations (ex.: two-state buttons for audio and canvas sessions, grouped buttons for brush and canvas) and with the platform due to Qt framework that always makes the interface's look native and its layouts that preserve OS

specific padding and position of widgets (see Fig.1 and Fig.2).

- The forgiving principle was followed by showing various dialogs in order to avoid accidentally made errors (ex.: before clearing the canvas or quitting) and by introducing the undo button that can restore the canvas.

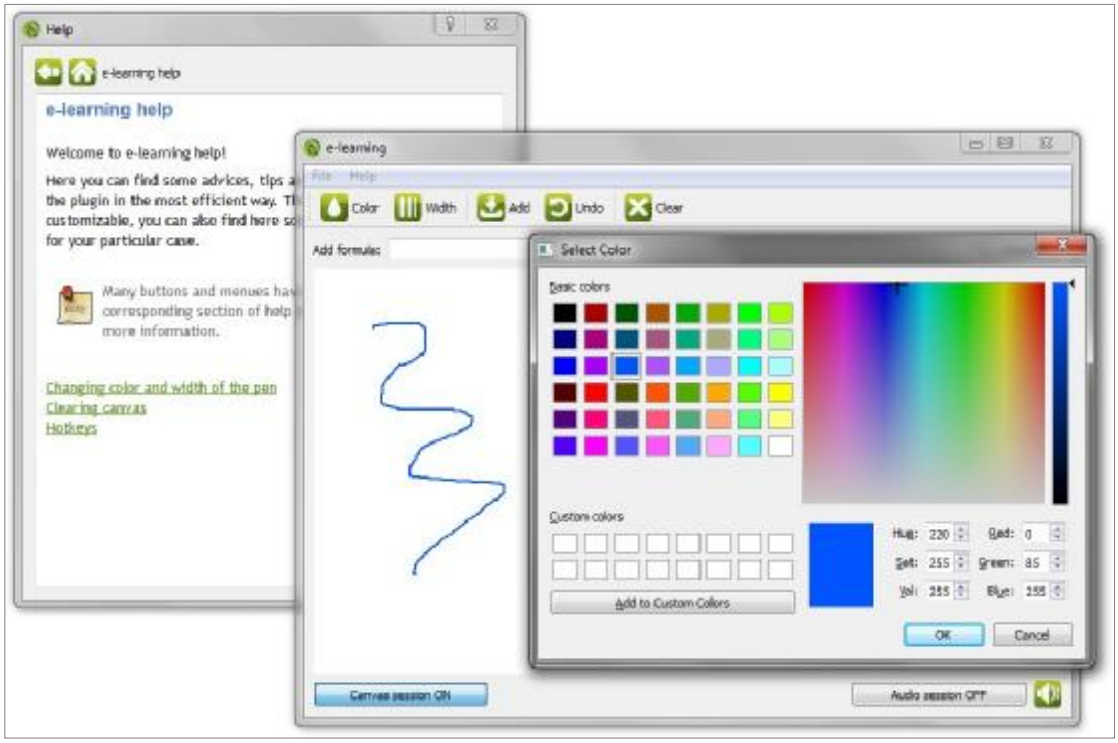

Fig.1: Developed interface on Windows 7 OS

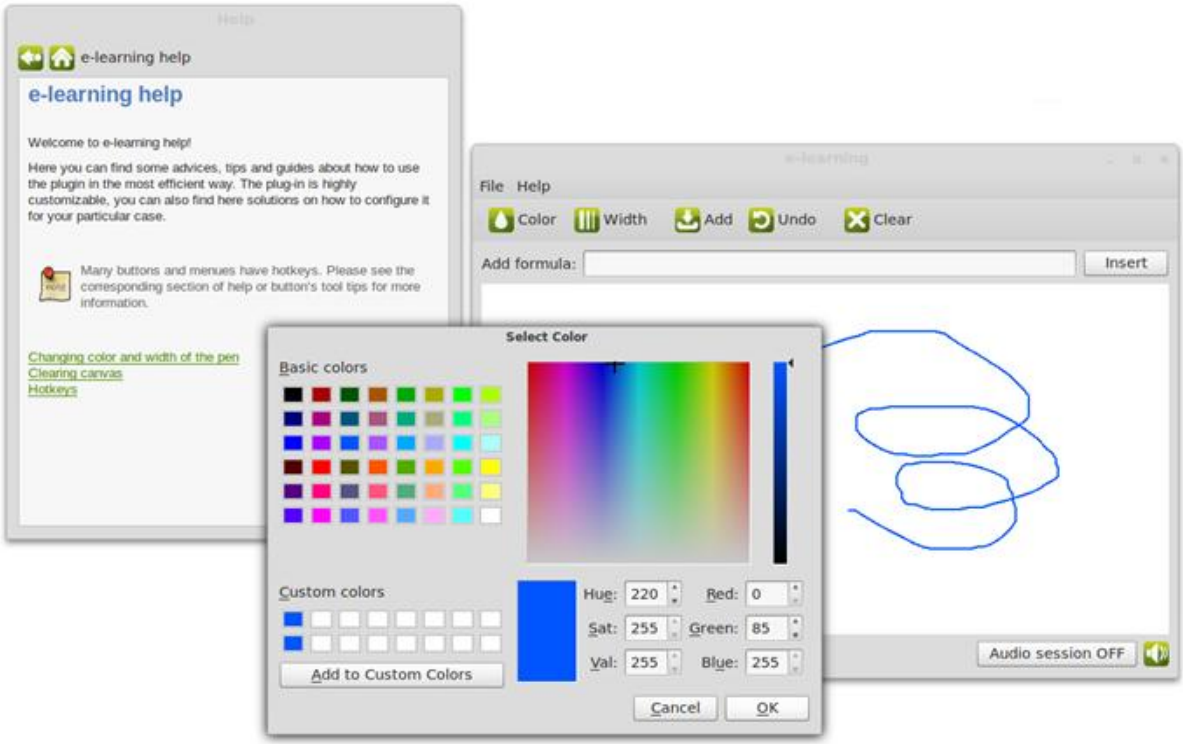

Fig.2: Developed interface on Linux OS (LinuxMint, Gnome 3)

VI. Conclusion

In this paper an introductory overview of user centered design of graphical user interfaces was provided. It has been shown that, due to the properly laid foundation, this approach to designing interfaces is superior to other possible approaches to this problem. A number of important criteria have been pointed out, against which the convenience and informational efficiency can be formally measured.

Nevertheless, it is crucial to realize that the paradigm of user centered interface design does not provide solutions by itself. In its essence, this methodology enumerates the features a certain concrete method must have in order allow the eventual arrival at a result with maximized level of usability. Thus, the creation of a concrete interface remains a rather complex task.

In conformance with the indications of user centered approach to GUI, the interface described in this paper was tested against the user interface stress test list constructed by the HCI team of the Technical University of Moldova [7]. The following result was achieved:

- the interface preserves the style of the current theme and OS as it was made from Qt's standard widgets for which such behavior is common;
- it is possible to navigate through the interface using both: only keyboard (each action has shortcut, the tabs are set in the right order: from top to bottom, from left to right) and only mouse;
- the interface looks well even having 125 dpi set because it uses relative sizing and Qt's layouts;
- minimal window size is set in the way that the interface fits even 1024x600 net-book screen;
- currently the application's interface is in English, the default language; it has also been localized to Russian. The preferred language for the user is detected and applied automatically to the GUI.

User centered design approach significantly improves the learning curve and usability of any programming product. The amount of time spent for mastering such software is reduced, it makes using the program intuitive, thus even in extreme situations it is impossible to make critical mistakes; and the most important is that such interfaces make users happy by using the software.

VII. References

1. Pidgin community, Pidgin, the universal chat client, http://pidgin.im/, 2012.
2. Norman D., Draper S., User Centered System Design: New Perspectives on Human-Computer Interaction, 1986.
3. Ebbinghaus H., Memory. A Contribution to Experimental Psychology, 1885.
4. Nielsen J., Progressive disclosure, http://www.useit.com/alertbox/progressive-disclosure.html, 2006.
5. Raskin J., Human Interface, The: New Directions for Designing Interactive Systems, 2000.
6. Nokia Corporation, Qt – Cross-platform application and UI framework, http://qt.nokia.com/, 2012.
7. HCI team of the Technical University of Moldova, UI stress test checklist, http://info.railean.net/index.php?title=UI_stress_test_checklist, 2011.