Title: Minimalistic user interfaces: the case for blind persons
Author: Michael Pozhidaev
Published: Proceedings of the 7th conference “Eastern partnership”
City: Przemysl, Poland
Date: September 7–15, 2013

Although widely popular graphical user interface (GUI) provides nearly all features users need for interaction with various types of computers, there are some cases when it is come out that GUI is not really the most suitable approach. Usually it is some kind of remote or embedded systems, terminals and so on, which have to follow some another conception due to performance or communication restrictions. We'd like to consider here one more case: minimalistic user interface designed for blind persons. Speaking in a few words what is a problem with GUI for blind persons, it is enough to mention that on the one hand GUI is designed to be controlled basically through mouse, which remains mostly inaccessible for blind users, and on the other hand GUI brings to user a lot of information usually turned out completely needless for a user without sight, but who still has to handle it during navigation procedures. We have no intentions to prove this statement, since it is quite obvious that GUI interaction takes a lot of extra time for navigation for blind users, and would like to propose real design of a minimalistic environment for blind users, which can be accepted as rather suitable for practice according to our experience and is chosen for implementation in Luwrain project.

First of all, we need to state what real requirements should be taken into account. These requirements could be itself a subject for research since it is not obvious what set of user interface elements is enough to cover all features, which system should provide for sufficient interaction. As it was shown earlier in [1] nearly all work object can be represented strictly in text form: text edit, lists, trees etc. Blind users can navigate over text blocks well, so, roughly speaking, applications should be a set of "areas" suitable for representation of text information. Although blind users gets any feedback through speech output, corresponding the on screen output still should be maintained properly since it is needed by users with partial sight. We describe there basically on screen visual content, implying speech output is also provided.

In the conception we propose entire screen is divided onto several areas shown in tiled mode. We will suggest corresponding structure and algorithm for handling area tiles, but now it is necessary to consider how applications can handle user commands. No problems to use general events approach to translate any user input as it is used in various GUI implementations, but there are special cases needed for user dialogs and undesirable operation interruption. Using the term “user dialog” we mean special text area shown on a screen as one function call which does not return until area is closed. Such dialogs can be used for user confirmations, files selections and so on. The calling procedure waits user decision and continues depending on user answer. Here is the list of all features which we consider enough to construct usual variety of things applications need for:

1. User interacts with a set of applications and he should be able to switch between each other quickly.

2. Each of the applications should consist of set of text areas shown on the screen in tiled mode.

3. Applications sshould be able to open dialogs areas as they are defined above.

4. User should be able to interrupt undesirable operation launched by an application.

5. There should be a method for quick information search in opened areas.

6. Applications should be able to operate in multithreaded mode.

Well, the implementation of several of these requirements is quite obvious, but for others we had to perform additional research and experiments. let take a look how to satisfy them. We should begin from execution threads model. The model of threads is related to events queue essentially. We propose there should be one and only one event queue in any given time as a pivot of a general design. Everything else revolves around it. This queue should accept two types of events: user input events and thread synchronization events. All of them have one common property: destination application or destination area they sent to. Evidently, the events of both types can be collected only in common threads-safe queue. Otherwise there can appear a race condition.

Moreover, input events source (the code for keyboard touching reading) must be executed in a separate thread. That must be done this way because user should have a method to cancel the operation he don't want to continue anymore. In this case he presses special keys combination to cancel entire the events reading thread and restarts it again.

Actually there is one another problem related to the events reading thread cancelling. Let take a look at user dialogs implementation. As it was mentioned above user dialog area is appeared inside of one special method call, but since events reading and events handling are performed in one single thread (multithread implementation is also possible but rather more complicated and significantly more system resource consuming), until this method is finished none of the other events can be processed. But events handling must be continued as it is needed for interaction with user dialog area itself. This problem can be easily solved by launching one more events reading and dispatching procedure until corresponding area is closed. It can be done with the same thread, no problems with that. But it causes one more difficult situation. Let user has many opened dialogs (and that implies he has many launched but paused procedures for events reading) and with this circumstances user suddenly decides to close one of the application, having opened dialog somewhere in the middle of the call stack. It is impossible as we are not able to remove anything from the call stack unless we cancel entire execution thread for events handling with all opened dialogs. It looks like a rather rough method but actually It is quite appropriate solution, and that gives one more reason to have two threads: one as events origin and another for events dispatching.

We'd like to suggest one more structure with corresponding algorithm for its processing within requirements above. We have mentioned text areas on the screen should be shown in tiled mode but strategy how to obtain exact areas positions and size remains still unclear. Entire tiles structure can be represented in binary tree. Each node of this tree can be one of the two types: nodes for vertical dividing and nodes for horizontal dividing. Each leaf in this tree is an area to show on the screen. Entire screen width and height are given, how to get convenient areas coordinates automatically? We propose the following procedure:

1. With each of the nodes (not leaves) special number must be associated: how many leaves are there under this node. These numbers should be calculated by obvious recursive procedure.

2. From the root node let do the following recursive procedure:

   (a) On each invocation the procedure receives node and rectangular part of the screen (on initial invocation the entire screen).

   (b) If received node is a leaf (area) it gets corresponding part of the screen.

(c) Otherwise received region is divided horizontally or vertically (according to node type). Portions of the parts after the dividing should be calculated proportionally to the numbers of each branch assigned on the first step of this algorithm. After that both branches must be processed with this procedure recursively.

This algorithm yields rather accurate dividing, but it can be improved by adding the processing of dialog areas as a special case.

References

[1] Pozhidaev M. Text-based environment for blind persons: conception and operating system design // International research journal, 2013, issue 2, pp. 63–66.