Introducing LUWRAIN: Can GNU/Linux help us rethink accessibility solutions for the blind?

Michael Pozhidaev
michael.pozhidaev@gmail.com

Linux Journal, 2015, issue 7
(Texus, US)

Michael Pozhidaev has been dreaming of changing the ways of accessible interaction for people with vision problems for more than ten years. Michael is 32 years old, has a PhD in computer sciences and is employed as a software architect at Electronic Moscow. He loves piano music and goes crazy when he gets a chance to take a flight on a big jet or visit a new airport. He welcomes messages sent to michael.pozhidaev@gmail.com.

When blind people work on personal computers, their mental perception of images replaces visual data on the screen. Users construct these images by getting portions of data transmitted from the machine in one of two alternative ways: either through the help of a speech synthesizer or through a braille display. The most popular solution to this kind of accessibility technology allows users to work in a typical graphical user interface (GUI) with the assistance of screen-reading software. The utilities, usually called screen readers, announce every step users perform by describing each change occurring on the screen as words.

This approach was a true breakthrough in the world of accessibility for blind people in information technologies. There have been a tremendous number of success stories of people with disabilities who are able to work, study and do a lot of other things completely on their own. Despite these wonderful benefits, there still are unresolved problems, mainly caused by the GUI itself. When using existing screen readers, the mental images should correctly conform to the rather complicated window environment. Children who are born blind are never able to see the sky and clouds. How can they understand a screen reader's feedback? The feedback always comes in terms of GUI elements, which always have a visual nature. Would blind people feel comfortable enough using this? Audible announcements of GUI changes on a computer might be a suitable solution if users had some prior experience in a windows system before losing their eyesight. Unfortunately, this is not always the case.

The second reason to look for an alternative solution is the fact that GUIs are convenient and effective only with a mouse. Unfortunately, a mouse typically is not used for work with screen readers. Blind users have to work with keyboards only, sometimes emulating mouse actions with keyboard commands.

1 The LUWRAIN Project

All of these factors make work a major struggle with a GUI, and research has to continue to discover new ways of interacting to improve the situation. Research efforts have yielded some new technologies, such as Adriane, but let's take a closer look at one more concept.

LUWRAIN is a GNU/Linux distribution in which the main user environment as an implementation on Java. Although its main part is suitable for running on any other operating system that has proper Java support, LUWRAIN launched in this way will be less functional than as a standalone OS.

LUWRAIN attempts to accomplish the following tasks:

- Making personal computers accessible to blind people who were unable to use them before due to various reasons.
- Making the environment of popular everyday applications more comfortable and effective for people who spend a lot of time working.
- Suggesting a solution for the problem of distributing accessible applications for blind people (it is exactly for this reason that LUWRAIN supports launch on any system with Java support).

LUWRAIN isn't a competitor to screen readers, because screen readers attempt to ensure that all things accessible for sighted users will be accessible for blind users as well, but without any care for practicality or convenience. On the contrary, LUWRAIN offers a solution for everyday work for a fixed number of tasks with maximum comfort and speed. If LUWRAIN's functionality is insufficient for a particular situation, you can install it just as an application and leave other tools untouched.

2 Text-Based Interface

LUWRAIN's main characteristic that distinguishes it from a screen reader's approach is its new type of interaction. If you want to make an environment comfortable and effective, you should simplify the image in the mind that reflects the working space on the screen. While working, a user's thoughts should be focused on the work itself, and needless interface elements should not distract the user. Regarding LUWRAIN's user environment, it would be fair to acknowledge that LUWRAIN adopts a number of ideas previously proposed in the Emacspeak add-on for the GNU Emacs text editor.

Let's take a closer look at the LUWRAIN interface. LUWRAIN rethinks the behavior of popular widgets in such a way that no visual data may be involved in their representation. A monitor still is used, but now it plays a supplementary role. Pictures on it help users with low vision, but there is nothing difficult if graphical information is totally unknown. Next, LUWRAIN also requires a redesign of general application structure, because all new controls must be organized in a completely different way from what people are used to seeing with a GUI.

Historically, when the GUI Emacs authors tried to implement that definitely were much larger than the bounds of the text editor (for example, mail reading, calendars, file

management and so on), they didn't realize what a brilliant idea they actually had. They clearly proved that nearly all objects can be represented in text form only. Indeed, roughly speaking, if you'd like to write somebody a message, you can ask one script to prepare a message template in a text file having a "To:" string on the first line, a "Subject:" on the second and the message body on the rest. After filling out all the necessary values in the offered template, you call another script, which looks at what you wrote, constructs a valid e-mail, and sends it through your favorite relay.

The text itself (as well as everything that can be described as text) is the most suitable data structure for blind people because they can explore it easily through speech or braille.

- When users jump from one line to another, they hear the line's text.
- When moving from letter to letter, users get the character under the new cursor position.

Several extra commands enhance navigation. For example, when jumping between words, users would "hear" the next word. This behavior always is the same, regardless of what screen-reading technology the users choose. The substantial difference is that in a GUI, this kind of navigation is used for text edits only (which are, apparently, just a subset of a large variety of existing controls). In Emacspeak and in LUWRAIN, it is used everywhere.

Besides generally improved accessibility with a ubiquitous text interface, users have some additional advantages. If the text is everywhere, users can do a text search in any place they want. Users can press the corresponding hotkey and type the corresponding substring. After that, the search either finds the substring or ensures that this string is absent. The second useful aspect is that in any arbitrary place, regardless of whether it is a file manager panel, a calendar page, an address book entry or a media player control, it is possible to copy the entire content as text to the clipboard — for example, for e-mailing to a friend.

It is important to be able to know what items are absent on the screen, because screen readers mostly take care of describing only the elements they recognize. Usually there is no way to ensure that something isn't shown on the screen, because users can't trust that screen readers understand the GUI completely. If everything is offered in text form, this is easy to do, because no problems arise when going through the text from top to bottom. In a GUI, on the other hand, it is much harder, because the focus doesn't need to stop at every element on the screen. Some of them can be skipped.

3 Constructing an Application for LUWRAIN

It is impractical to try to access an entire application's functionality in one solid text space, so usually it is recommended to split an application's objects into several parts called areas. Each area implement the text-interface behavior. Their exact setting is dependent on the purpose of the application and should be designed very carefully.

If you consider a double-sided file manager, you can see that it requires three independent areas: two panels for directory browsing (left and right) and one more for listing continuous user actions (copying, moving or deleting). All of them behave independently,which means they have their own cursor, and changes in one of them do not influence the others.

All of them are shown on the screen in such a way that allows filling the entire screen space (as tiles). Two panels are positioned as usual, and the action list goes to the bottom. Once again, appearance on the screen no longer plays a substantial role; it is just a complementary source of information for people with low vision. But as the screen still exists, let me mention a few additional details about it. For better adjustment for special needs, it requires some new features. Each area on the screen is filled only with text data drawn with a monospaced font. Users can choose the color scheme and the font size freely during work without any support by the application.

Each of the three areas, which the file manager consists of, behaves as a list. However, no matter what they are, they should be accessible as text. Each particular item is on a separate line; the movements are announced with some supplementary information, such as whether the current item is a directory or a regular file. All extra comments can be switched off if the user holds the Ctrl button on the keyboard while jumping from line to line. This little trick significantly increases efficiency, but it is not the only example in the file manager. LUWRAIN has such tricks in various places everywhere, and that makes the environment truly more suitable for blind people compared with screen readers, which sometimes are not able to distinguish the main content from the decoration.

4 LUWRAIN Widgets

LUWRAIN has its own library of classes that are useful for constructing an interface with the controls. Generally it is similar to the usual GUI widgets, but redesigned in the ways described previously. This library includes lists, trees, text inputs, forms, menus, some special items like calendars and so on. Of these, the forms are probably the most difficult. The forms in LUWRAIN are text areas containing one particular control on each line. Lines begin with a control name and can represent a text input, a yes/no check box, an embedded list and so on. Lists always take a single line while a selection is always performed in a separate area.

In contrast to GNU Emacs, LUWRAIN doesn't have real buffers with real text content for all objects. Text representation is there just as some sort of intermediate level, obscuring internal structures. The internal structures can be very complicated without any restrictions, and their logical level makes them easily observable by a user. You can design your own approach on how to handle any new nontrivial data model in a text way, creating a new custom control.

5 The Environment Internals

Behind a rather simple front-end conception, there is a complete UI infrastructure

based on events dispatching and handling with the support of multithreading and user dialogs (like some type of modal windows). As I already mentioned, the main language chosen for LUWRAIN is Java. Discussing all reasons for this decision here is almost pointless, as the pros and cons for any particular language are very arguable. But, one thing plays a crucial role — the substantial number of existing libraries created for Java developers, the most important being JavaMail, Rome (the RSS parser), Apache POI (office document filters) and many others. The LUWRAIN core translates various input commands and internal environment actions to events that pass through the main loop queue and, after that, are dispatched to the corresponding object, considered to be the most suitable destination.

The events also are useful for multithreading synchronization. The application may issue as many threads as it wants, although only one of them is allowed to perform operations with the user environment. For this purpose, LUWRAIN has a corresponding type of events that safely allow data exchange between the background threads and user interface.

6 Pop-up Areas and Applications

The main loop procedure is capable of the recursive calls needed for the appearance of dialog areas (modal windows). These areas can be shown as a single method call placed inside some other wrapping handler. An existing unfinished upper handler freezes the top-level event loop, and that potentially could freeze the entire user environment, but the recursive loop instances solve this problem.

A user gets the dialog areas along one side of the screen. Regular applications may open them only along the bottom, but LUWRAIN opens its main menu on the left (yes, LUWRAIN has a main menu, which can be opened by the corresponding button on the keyboard where everybody expects it to be), and on the right, there is the context menu with the actions associated with the focused object. Most actions in the main menu are accessible through short commands, which can be issued from the command line. It takes the system-wide commands only, which should be applicable regardless of what applications are currently opened. Having this additional input method increases speed, because in a noisy environment — say, in an airport or aircraft — it is easier to type "message" quickly than to select menu items by listening their names.

Applications in LUWRAIN always fill the entire screen space. Only the amount of system resources limits the number of concurrently launched instances. Users easily can switch between applications by pressing Alt-Tab. (Actually, the term "application" isn't the most proper one, as LUWRAIN applications share the same memory space inside the Java virtual machine and don't go to separate processes, but it's the easiest term to describe them.) The LUWRAIN distribution includes several standard applications, such as a double-sided file manager, mail client, feeds reader, music player, command line to launch bash expressions (implemented through Java Native Interface), ad-

dress book, personal scheduler, office document viewer and editor, notepad and others.

7 GNU/Linux Environment

Now you know enough about everything that happens inside the Java virtual machine, but there are a lot of external things that revolve around it. LUWRAIN should interact with system level services for managing network interfaces, attaching/detaching removable media and for other similar tasks. The most convenient way to do that is, evidently, D-Bus. D-Bus is accessible from Java, and a number of existing D-Bus services cover a lot of needed features (for example, you can use Network Manager for network configuring). This means LUWRAIN will be consistent, as more services are introduced to D-Bus. If one day Lennart Poettering and the community around him get system in a commonly acceptable state that no longer raises any concerns, that will add the final missing piece to a nice design of LUWRAIN as a complete operating system.

LUWRAIN is capable of being installed by blind people freely without any sighted help. This is achieved by cloning a live CD system to the hard drive and can be done rather quickly.

The LUWRAIN story would be really nice if all things could be executed inside the Java virtual machine, but unfortunately, that's not the case. Let's step back a bit and think about whether LUWRAIN is applicable for all commonly required tasks. It is not necessary to consider The GIMP or various types of CAD software, because the nature of such programs requires eyesight, regardless of what interface is offered for them. However, there are a couple cases that are exceptions for LUWRAIN. These are Web browsing and some closed applications (the most popular example of this is Skype). You can't complete these tasks in Java with a text-based interface, so there needs to be some alternative solutions.

Both Mozilla Firefox and Chromium support accessibility features for blind people but in different ways. Chromium does it through its Google ChromeVox extension, which can work independently from any other GNU/Linux assistive infrastructure. I don't discuss it in detail here because it just works. Firefox does it through access to the so-called AT-SPI (Assistive Technology - Service Provider Interface). AT-SPI initially was a part of the GNOME Accessibility Project and allows gathering all information of launched applications and their GUI elements into a single place. After collecting this information, it can be passed to some assistive technology that transforms it into speech and provides it to the user. Usually this job is done by the Orca screen reader, but Orca itself is strongly linked to the GNOME environment and isn't used for any LUWRAIN tasks. LUWRAIN includes a special tool to replace Orca that is intended for collecting AT-SPI information and translating it to a human-understandable form. (Great thanks to Mike Gorse, who always is ready to help and clarify details of AT-SPI behavior!) The nature of basic Skype operations allows their transformation to a text-based interface, but it is closed software and LUWRAIN is unable

to do anything with it. In the meantime, the GNU/Linux version's UI is based on Qt, which also is supported by the AT-SPI functions. All that is needed is to compile and install the special qt-at-spi bridge.

No doubt, it would be better not to have any exceptions for LUWRAIN's text-based interface proposal, but we should be happy that it is possible to overcome such special cases in an appropriate way.

Speaking of the LUWRAIN details that work outside the Java virtual machine, let me point out that all tools launched in the X.org server (LUWRAIN's main window, a Web browser, Skype) are under control of the special tiny window manager (of course, not written from scratch).

Conclusion

In concluding this LUWRAIN technical overview, it is interesting to consider what LUWRAIN can and cannot change in the lives of potential users, although this project is still in a rather early stage. How will users possibly perceive of this system, which has many significant differences compared with most existing solutions? What new areas could it open in the technical world?

First and most important, LUWRAIN is aimed to be useful for users who previously were unable to interact with PCs due to various reasons. It might be appreciated by seniors for general operations, because LUWRAIN requires a minimal amount of technical knowledge. Its interface is constructed in a way that makes usage possible for most users after following the short system guide offered for everybody on booting the live CD. Experienced users likely will be able to work without any prior learning at all (according to a series of our small experiments).

However, some things still may raise concerns. The environment is "in itself", which means that creating a new application will not make it available for blind users by default. Also remember that LUWRAIN doesn't imply the exclusive selection of a platform to work. Users can have several machines for different tasks. On a desktop, users can continue using a particular required application, but on a laptop, LUWRAIN could offer a better environment than any other OS. Also, if people want to use LUWRAIN due to its better organization, but aren't able to abandon Microsoft Windows or Mac OS X (for instance, because other family members use it), they still can install LUWRAIN on any OS with proper Java SE support. The same is true if users are unable to switch to GNU/Linux due to some specific software.

Let me also note that LUWRAIN allows using bash expressions. Does anyone refuse to ping just because it is a command-line tool? The command line itself is one of the most accessible conceptions for blind people.

Because LUWRAIN provides an easily understandable interface for blind users (hopefully) and potentially could launch on any OS with Java SE support, it could be considered as a platform for distributing accessible speech-enabled applications. Perhaps there are vendors who would like to have an application that will be accessible

reliably to blind people. For example, it could be used for payment systems, social applications for health care purposes and so on. With LUWRAIN, vendors can do this without any prior experience in creating accessible tools, because it just takes already prepared Java classes. LUWRAIN not only promises that it will be understandable for blind users, but in addition, it offers a platform for launching a newly created application, which is free of charge, eliminating barriers to software costs.