The Framework for Accessible Applications: Text-Based Case for Blind People

Michael Pozhidaev
National Research Tomsk State University, Computer Science Department
Lenina Avenue, 36, 634050
Tomsk, Russia
msp@altlinux.org

ABSTRACT

This paper offers a Java framework for creating accessible applications for blind and visually impaired people as part of a proposed general conception based on the maximum use of objects filled with text data only. It offers new types of applications more easily recognizable by disabled persons, helping them to do their work faster and more comfortably. Strong and weak points are analyzed. The published prototype of the proposed platform is described as well as the conclusions of the performed experiments. The prototype is implemented on Java SE and wrapped by a GNU/Linux environment as a bootable ISO-image.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*User interfaces*

General Terms

Experimentation

Keywords

accessibility, blind people, user interfaces, usability, Java, API

1. INTRODUCTION

During last decade we have seen the process of diversification of information technologies which people use in their everyday lives. If during the mid 2000-s almost all tasks were performed using Microsoft Windows and desktops only, today's solutions are very different, regardless of whether we are speaking about software platforms (Mac OS X, iOs, Android are have a wider distribution and very often can be considered as real competitors to Windows) or about ways of interaction with PC's or gadgets (multitouch as a replacement for the mouse). In the meantime, blind and visually impaired people (from here on in "blind people") typically still are not able to enjoy all the advantages of this process, mostly because the accessibility technologies are just addons to interfaces initially designed for sighted users. This kind of solution can be considered only a partial solution since it offers the general ability to do the same things as everybody but sacrifices the efficiency and level of personal comfort.

Each time it is necessary to create an application designed and highly accessible for blind people developers always create a usual application for graphical user interface (GUI). Their main distinguishing characteristics aren't more than features to enable the ability to contrast colours and adjust font sizes. These functions are only measures for users with partial sight. If somebody has never dealt with GUI as a sighted user the complete and proper understanding is generally impossible (especially for senior users). Respecting the increasing requirements for an accessible environment we would like to propose a platform designed completely for blind people which could be a stable solution for the problems mentioned above.

Speed, comfort and good understandability usually mean just one thing; keeping things simple. The question is what kind of user interface (UI) could be simple and at the same time functionally sufficient. The suggested solution to an interface for blind people is, on one hand, simple enough and, on the other hand, could provide the replacement for most GUI widgets. We will discuss the pros and cons, and discover how visualization and speech output can mutually supplement each other. The prototype of the proposed platform was published to demonstrate implementation on Java. It consists of the core maintaining new interface and launched applications, as well as a set of Java classes making API for creating new applications. This environment could be launched on any OS with the Java virtual machine but we would like to see it as a complete OS on the Linux kernel (published prototypes are prepared in this way).

2. TEXT-BASED ENVIRONMENT FUNDAMENTALS

2.1 General conception

First of all, we would like to describe an alternative solution to ensure it is really suitable and comfortable the blind. There is some prior experience in this area described below and we would like to respect it. The new environment enable speech output and some visual representation without involving any screen reading software. The picture on the screen remains very important because it is used

by users with low vision, though it plays only a very supplementary role now. We suggest splitting the screen into several rectangular areas, and filling its entire space with some tiles. Each tile shows some textual data related to the corresponding object and is closely associated with speech output. Font size and colour should be easily adjustable to suit a particular user's requirements.

The method of how to constructing a comfortable and sufficient speech representation for all objects needed for work is one of the main goals of our research. It is described in detail below but, roughly speaking, we are considering all of the usual GUI widgets in a way where no graphical and visual data is involved in their representation. Fortunately, there are only a few cases when it is impossible (e.g. the web page cannot be described in words without lost of entirety). Let's consider, for example, a text edit box: there is nothing graphical about it, but the selection of some part of the text with a mouse is performed through choosing some other colour and this happens by using visual information. Hence, we must invent something else for text selection but there is no need to create anything new for text input itself.

The environment is designed for distribution with some standard set of applications prepared and organized in a new way. At first glance it seems that we will face a tremendous number of new functions to implement, but actually there is a wide range of libraries for Java, distributed under public licenses; so there is no need to do anything from scratch. We should simply prepare a new interface for them with our new platform.

2.2 Known experience

Usually blind people use screen readers launched on a GUI, but we will not review them as we are interested only in software designed as "audio desktops". There are currently three solutions which can be considered as providing a speech environment without screen reading. All of them accomplish this in different ways. They are Emacspeak [20], Dolphin Guide [2] and Adriane [8]. Emacspeak certainly had a major impact on the outcome of this paper. It is an add-on for the popular text editor GNU Emacs [22] which has a lot of additional features covering areas significantly wider than just usual text editing, such as file management, mail reading, calendar etc. The main advantage of Emacspeak is that with some proper training it can help users to be highly efficient(almost to the level of a sighted user). We are wanted to keep all of the positive parts of that interesting experience. However, unfortunately, it has a lot of restrictions existing mostly as a consequence of its add-on nature (GNU Emacs has nothing close to such entity as application). All weak points of emacspeak have been analyzed in our previous publication [18] in detail.

Dolphine Guide is a high-level screen reader with a lot of additional information about user interface for a fixed number of particular applications. It obscures interaction with these applications and replaces it with its own very user-friendly environment. Although Dolphin Guide is able to be a solution for inexperienced users it cannot be considered as a flexible environment providing efficiency for users because it handles only a fixed number of use cases. In addition, Dolphin Guide needs a complete OS (Microsoft Windows) to work.

These problems are also inherent in Adriane except that it is based on a Gnu/Linux distribution. It involves a number of applications for various tasks which were wrapped with speech-enabled interactions. Adriane cannot be considered as a platform for constructing new accessible applications because it hasn't any strict and flexible core providing API for developers (although its interface could be considered as quite consistent) and the components it uses are mostly based on the Dialog [23] utility and Bash scripts [14].

2.3 Pros and Cons

Let us consider all the reasons why our application makes sense and look at arguments for why it could have some weak characteristics. To respond to criticism, which we may expect, we will give additional comments for why we don't take them seriously and don't think these criticisms are crucial. As a stronger approach, everything is written like it would be as a complete OS, and we will also look at why the case of running in an environment on Microsoft Windows is significantly weaker.

The statements supporting our proposal could be as follows:

1. Users no longer need to struggle with GUI and that saves a lot of time, increasing efficiency. GUI is a very unsuitable solution because in complicated cases blind users should reference a screen structure which they don't see. On desktops and laptops, GUI is useful only with a mouse, which is inaccessible for blind users. With tablet computers, where multitouch is popular, blind users should be able to touch the screen in same position multiple times. That is easy to do having visual information, but in our speech-enabled case it turns out as a very time consuming procedure. In the meantime, the presence of GUI is a completely artificial problem because GUI is created by the hands of people and isn't something essential for interaction with a PC.

2. A new type of interface could be easier to understand for blind users with a lack of experience with computers, especially seniors. With GUI it is necessary to have a proper understanding of what it is. That is not a problem for users who have previously used a PC as a sighted user. But for those who have never seen a computer screen such understanding becomes a really serious problem.

3. The general conception and suggested implementation could be a platform useful on mobile laptops as well as on embedded devices based on the ARM platform [24]. In conjunction with previous point this could have social value as it would now give access to a wide range of digital services for disabled people. This makes it a solution with social value.

4. For experienced users there are certain tasks that can be uncovered with existing solutions. Usually this is related to speedy software development, preparing materials with Latex [11] or Lilypond [7]. A new approach could be well adjusted for these tasks.

Criticism of our proposal could be as follows:

1. It is an environment "in itself". This makes blind people more isolated from society and blocks their access to software not included in the new system.

2. This proposal requires creating some number of already existing applications, e. g. for mail reading, news reading etc.

3. Not all applications could be reconsidered with the proposed conception. For example, a web-browser could be accessible only with the screen reading approach.

Our responses to such criticism would be as follows:

1. About isolation:

   - Our conception is best suited to be a user accompaniment preferably on mobile computers, while existing desktop systems remain available and everybody is still able to use a general purpose OS like Microsoft Windows or Apple Moc OS X on it. There is no need to have a specific single universal computer (in contrast with mobile phone). If it is launched on Microsoft Windows. Java will be able to make this happens.
   - The system isn't totally "in itself". It can provide access to command line utilities through Bash or some other shell. In fact, command line utilities are one of the most accessible ways for interaction for blind users. Although, of course, it is suited mostly for experienced users. In addition the command line is needed for running the utilities usually launched exactly in this way (e. g. "ping").
   - There is a significant gap between potentially possible and really available features for blind users. With GUI we may think that they are not isolated but it doesn't mean that we are really able to do everything that is needed.

2. About creating new software:

   - There is no need to develop anything from scratch. We are working in the area of Free and Open source software, so we can use a lot of libraries already created for Java for our implementation.
   - According to the Pareto rule [9], 80% of software needs are covered by approximately 20% of the features in the software. Therefore, we may expect that there is some appropriate level of functionality which would be sufficient for most tasks.

3. About exceptions: yes, there are some exceptions, but we can include their workarounds into our system as exceptions. Speaking about the web-browser, we can take Chromium [12] with the ChromeVox extension [19] which will run outside of our environment and that will not bring any inconvenience to the user. Generally, the complete OS should be some sort of "hybrid" system if it is based on the proposed platform. Some popular applications, like photo editing or computer aided design will never be required because they are pointless without visual information.

3. THE FRAMEWORK DESCRIPTION AND LUWRAIN PROJECT

In this section we will consider all of the valuable details of the Luwrain project which we proposed as an implementation of the conception and the framework described above. Although Luwrain includes some research and experimental tasks we intended to get a stable product and fully functional OS based on it suitable for developers as well as for a wide range of consumers with sight restrictions. Anybody who is interested in getting a complete understanding of what Luwrain is and how it works is welcomed to try current prototypes freely published on the corresponding website at http://luwrain.org/.

We guess that it is necessary to describe this work from various points of view as we are speaking about not just a theory but also about an exact technical approach. Most things described in the sections below should be reflected in the Luwrain API as a set of Java classes. The Luwrain classes provide various levels of customization. For example, the class for list view needs only a set of items and does all tasks according to the conception. However, if a developer would like to prepare his/her own controls, he/she has everything needed to do that. Each application for Luwrain should be distributed in the form of .jar file as is usual for Java libraries.

3.1 Accessible controls

The main requirement imposed on a set of controls is to have functionality equal to that of GUI. Of course, except obviously, the requirement of being fully accessible for blind users. Speaking about a "control" we mean here items such as text edits, list views, menus, tables, forms etc. Forms include various things, like edits, check boxes, some custom controls and so on. Every class of a control object should generate output for the screen and for speech simultaneously and in such way that output for speech should be fully sufficient for any kind of work while output for the screen plays only a supplementary role. One additional requirement for any type of a control is providing access to any part of the object without potentially inaccessible information. This is usually very relevant for exploring the spelling of any string in a letter-by-letter manner. A lot of users are unable to completely rely on the pronunciation of a speech synthesizer.

Text edits. The text edits (both single-line and multiline) speak the letter under the cursor on left-right movements and speak the line holding the cursor on up-down movements. On typing any letter, the letter should also be spoken. The important question is the selection of some text fragment. We guess that the most convenient way is to set a special point under the current cursor position to mark the start of the region, then go to some other position marking the end of the region and do one of the required operations (copy, cut or delete). It is a good idea to pronounce a text fragment being cut, copied, deleted or pasted. On reaching the bounds of a text area, a corresponding notification should be issued.

List views and menus. All types of items of enumeration should have a cursor marking not only a particular line, but also being free to point to any character on this

line. This is necessary as mentioned before for exploring the spelling of the item. On up-down movements a new item of text should be spoken and the cursor should go to the beginning of the new line. Every line should always have one additional empty line (otherwise, how could we know the text of an item if it is single and there is no way to go up-down?).

Tree view. Tree views are also possible. We can treat them as some sort of extension of a list view. If some particular item has children, it gets a plus or a minus sign for screen representation and a corresponding speech notification is added. Pressing the enter button on such an item consistently expands or collapses its subitems. The level of the item can be reflected by the indentation on the screen and corresponding speech suffixes or prefixes.

Forms. Forms imply a set of various controls such as text edits, check boxes, drop down lists etc. There are no problems with them if the constructed forms place each control on a separate line and adds a corresponding text prefix designating the name of the control. All lists should be drop down and the item selection should be carried out through additional popup areas (see below). There is one noticeable limitation: each form can contain only one multiline edit and it always should be placed at the bottom of the form filling the entire space below all other controls. For example, such an approach is selected for an area with the purpose of composing a mail message. It has a recipient address on the first line, and then the subject on second, some additional fields, but below all of them there is a multiline edit for message text.

We described some the most important controls for illustration but, of course, not all of them. Others can be reconsidered in the same way as these.

There is one rather serious problem often faced during work on accessibility technologies: there is an unpredictable amount of information needed by a user in different situations. When a user explores a structure of a tree view he/she should get as much information as possible, but if he/she looks for some particular known item only the name is required to be spoken. We suggest to use alt keys like Ctrl or Alt on the keyboard to switch the mode of output. For example, holding the Ctrl key always skips all supplementary information, saving only the items names.

We should mention one additional trick usually considered as rather popular and useful. Whereas all controls (lists, texts, trees or forms can be represented in text form in one or an other way, a feature to quickly search some text substring can significantly increase efficiency. For forms no matter whether this substring appears in editable areas or just in control names, it should be, so that users easily understand what it is by themselves.

3.2 Applications and tiling

Each application in Luwrain gathers several controls. Their number and types are defined completely by the purpose of the application. For example, a mail reader should consist of three controls: tree view with mail groups (Inbox, Sent Items, etc), the list of mail messages inside a particular group and the text of a particular message. On the screen they should be placed in the same way as in a usual GUI mail client but for users who work through speech only there shouldn't be any association between the objects location on the screen and their behaviour. He/she just has to mind that there are three objects and it has the proper way to switch between them.

That could be easily achieved if the environment implementation takes complete care of the calculation of a control's position on the screen. We can even think that there is no need to have a way to choose the position manually, it is enough to have a sufficient algorithm giving a suited position automatically in most cases. For that purpose we would like to suggest one of them. It would take a tree of tiles, each node of it has references to the two children and a boolean attribute whether this node implies dividing ina horizontal or vertical way. Given screen width and height, it calculates the position of each tile on the screen.

1. Performing Depth-first search on a tiles tree and calculating how many leaves has each branch of each node.
2. As a recursive procedure, do the following steps:
   (a) Call the procedure providing the screen width and height as well as the root of the tree.
   (b) Indicate if the provided node is a leaf assigned to it received screen position.
   (c) Perform the dividing of a received screen area into two parts in proportion, how many leaves there are under each branch, and handle dividing direction (horizontally or vertically). After that perform a call of the procedure for each branch providing the obtained positions.

According to our experience, this procedure yields rather good dividing for each application. It is necessary to describe what an application is in Luwrain design. First of all, we would like to note that the term "application" doesn't reflect the exact nature of the implied object. Very likely it would be better to call it an "applet" or "add-on" because Luwrain applications are executed in the same process as the environment itself (although they are able to initiate separate threads) and share with the environment the same memory address space.

The applications are Java classes where objects are registered in the corresponding manager. There could be multiple instances of a particular application (e. g., user can launch several file managers) and in addition there is a special technique to prevent some others applications from being launched twice (e. g., multiple copies of mail fetching application are pointless). In each case, currently only one application can be shown on screen and it is the one which is considered to be active. We are thinking about a special type of application visible permanently (e. g., for displaying news feeds) but we still are not sure whether it is really necessary or not. Switching between applications is performed easily and quickly with the Alt+Tab key combination.

With Luwrain distribution comes a set of standard applications. They are: double-sided file manager, extendable text editor, mail reader, news reader, terminal, media player, the application for office documents preview, personal scheduler, calendar, address book etc. Some sort of applications can be provided in the form of extensions and we would like

to see them as a part of community-driven activity. They are mostly clients for popular websites, like Twitter, Yandex and Google services, payment systems, social networks etc.

It is necessary to especially emphasize the question of clients for digital government services. Their presence in any assistive technologies could have significant social value. The question of an accessible alternative for office applications remains very arguable. Office document exchange is very active and there is no doubt that corresponding applications are needed, but full functionality with Microsoft Office is needless. With Luwrain we intend to provide such tools, but the set of their features is defined by corresponding Java libraries (see below). Office document representation in text form should be worked out, but full page rendering (e. g., for printing) apparently remains impossible. As a partial measure we suggest to use non-wysiwyg alternatives, such as Latex, since they are highly accessible, although they require some training and experience.

3.3 Events dispatching and popup areas

The various event dispatching techniques, very likely, are an essential part of any UI implementation and Luwrain isn't an exception in this sense. Luwrain has several types of events with corresponding rules of their routing. They bring information about user actions, notifications about changes in the environment and do multithreading synchronization. The last is a very important feature, Luwrain allows application developers to initiate as many execution threads as they need, but all interaction with the Luwrain core should be done in a multithreading-safe way and Luwrain provides some features for that.

Actually, there is nothing to describe in detail except one thing: our environment has special types of areas which applications are able to show. They are the so called "popup" areas. Their main distinguishing characteristics is that they can be shown as one method call, which ends only on the closing of a corresponding area. Since the environment carries out in one single thread, this causes some difficulties because the popup method call (usually placed in some event handling code) freezes the entire event loop execution. This problem is solved by implementing multiple event loop instances. The first of them is a main environment event loop and each new loop is launched for every new popup area. We can think of it as a very usual approach in UI design.

With popup areas we can show various dialogs and menus, continuing the execution depending on user choice. For example, a couple of popup areas have system-wide meaning. The first of them is the main menu which is an idea very close to the "Start" menu in Microsoft Windows. The other one is a command line always accessible with the Alt+X key combination. With this prompt a user is able to launch a particular application or do some action with system-wide meaning. This feature is very useful when it is necessary to do something in a noisy environment. For example, in some cases it is easier to press the Alt+x and type "mail" than to open the main menu and listen to its items. This idea of Alt+X command prompt was adopted from Gnu Emacs but with some modifications.

3.4 Why Java?

Luwrain is implemented mostly on Java. Its environment is executed completely inside of the Java virtual machine. The main reasons why we use Java is the large variety of existing Java libraries and Java is currently a common language for any kind of framework and platform. Speaking about libraries, for instance, if we are creating a mail client it isn't necessary to write all the protocol parsers, anyone can just use Javamail library [4]. Actually the number of involved libraries is relatively large, so we use Apache POI [1] for office documents format processing (exactly a functionality of this library defines how good documents support could be), Rome [3] for RSS parsing, and many others.

But the reasons aren't limited only because of libraries. Java has a rather stable API which is changed very carefully and moderately (thoughtless API changing in our opinion is one of the biggest problems in the world of Open source libraries). In addition, [15] the current Java speed of execution is comparable to the speed of C++ and that is a rather good result (sometimes benchmark resources offers information that Java gives 80% overhead over C++ time).

Some questions related to Java remain unsolved. There are some legal concerns, we can see that corporations can sue developers over Java [6]. Next, it is unclear if we could build something on Dalvik (or on coming ART) [5] which looks more efficient for ARM devices. Hopefully, these questions could be solved in future.

3.5 System-level services

If we are speaking about a complete OS, we should keep in mind various services for maintaining network connections and other system tasks, as well as the way of interaction between these services and the UI inside of the Java virtual machine. Current experience in the GNU/Linux world demonstrates a tendency towards D-Bus [13] as a tool for interprocess communication (IPC). Java has a corresponding interface to use D-Bus as well, therefore, we should just choose projects which provide the necessary functionality with the D-Bus interface. Fortunately, they are present for almost all tasks:

1. Network manager [25] for manipulating network connections
2. Udisks [26] for removable media management
3. VoiceMan [17] for speech output

Network Manager and Udisks are well-known projects, speech server VoiceMan has been developed earlier as a part of the Luwrain project, but it is implemented as a system service on C++. Currently VoiceMan takes text to speach through the inet socket, but it is just a temporary measure.

An actively discussed idea of the Systemd service [16] proposed by Lenard Poettering potentially could be nicely integrated into the system we are discussing. Installation on a hard drive can be performed by a blind person without any sighted help using the live system cloning technique [10] The main window of the Java environment is shown with X.org server [21] using a custom lightweight window manager.

If Luwrain is launched on Microsoft Windows these features will be inaccessible or redirected to corresponding Windows components.

4. TESTING AND RESULTS

The experience we already have consists of two parts: a general conception of testing done in the Emacspeak environment and user feedback collected on the publication of the first Luwrain prototype.

Using the proposed conception as it was implemented in Emacspeak was really successful (the author of this paper successfully graduated university and has done his Ph.D. thesis using it on a machine). The efficiency is really high and handles some tricky operations, like an installation without sighted help. In the meantime, it is still difficult to share this experience with other people because using GNU Emacs and Emacspeak requires a lot of technical knowledge. Some design problems of GNU Emacs (e. g., absence of an application) don't allow us to consider it as a platform satisfying modern trends for popular products.

The first Luwrain prototype published in the form of a bootable ISO-image was presented on March 1st, 2014. Feedback was obtained from two categories of users: from newbies and from experienced users. The feedback from newbies is more valuable because our system should match the expectations of wide range of users. With very short prior instructions users easily understood what they should do if they would reach some particular position as well as to use some application or open an object. In the future these instructions can be offered as brief guide on system startup. Experienced users wanted to treat the potential success or failure of the system as being highly dependent on the ability to get a first stable release, because this work obviously requires a lot of development resources.

After discussions in foreign communities, the authors may expect that there could be some interest in solving accessibility problems specifically through such specialized environments.

5. CONCLUSIONS

We have described all the basic questions related to a special framework for developing applications for blind and visually impaired people. If this product could be implemented completely, blind users would get a free tool for most of their everyday operations. With it they could read and write mail, track news, listen to music and books, etc. All of these things could become available easily. We have omitted the details of possible user perception because with this paper the description is focused on Luwrain as a framework for creating accessible applications.

The suggested approach should be considered very carefully because accessibility technologies always carry some difficulties, some amount of things impossible totally. This fact is quite obvious, for example, when we see that there are no technologies (and very likely will not appear for observable future) which could be able to describe in words any given picture. Blind people never will be able to do computer aided design and some other tasks in a visual nature. We should look for sufficient solutions, but not perfect ones. With this reality, the project authors are strongly convinced the proposed conception could solve some rather tough tasks. To obtain it we spent more than ten years in research including a large number of experiments.

It is necessary to properly discuss our arguments against GUI. A text-based environment should not be taken as a competitor to GUI. In some circumstances the existing screen reading solutions for Microsoft Windows, GNOME or Mac OS X can be the only possible solution. For example, if somebody must use the same applications as everybody in school or university, then this would be the case. Surely, the solutions in style of audio desktops should be aimed at the sectors left uncovered with screen readers and solve tasks remained unsolved by screen readers.

We are still speaking about work which is just at the phase of representing first prototypes. And the number one goal is to get it finished. The real value of this conception can be measured only by the probability of doing that. If we are able to solve some technical questions, other things, related to the available models for such work, still remain obscured. On the one hand, this project is Free and Open source, meaning it is non-profitable, on the other hand, it could have some social value. The question whether there could be development models, more suitable for this combination, or not remain opened.

6. REFERENCES

[1] D. V. Ajay Vohra. Use jakarta poi to generate excel spreadsheets from xml documents. http://www.javaworld.com/article/2076189/enterprise-java/book-excerpt–converting-xml-to-spreadsheet–and-vice-versa.html, 2006. Retrieved May 27 2014.

[2] Dolphin Computer Access Ltd. Dolphin guide. http://www.yourdolphin.com/productdetail.asp?id=30, 2014. Retrieved May 27 2014.

[3] M. Fortner. All roads lead to rome. http://www.jroller.com/phidias/entry/all_roads_lead_to_rome, 2009. Retrieved May 27 2014.

[4] E. R. Harold. *JavaMail API*. O'Reilly Media, 2013.

[5] D. Helleberg. Android internals: Art in practice. http://blog.dominik-helleberg.de/2014/01/28/android-internals-art-in-practice/, 2014. Retrieved May 27 2014.

[6] B. Kendall. Oracle wins ruling in case against google over java. http://online.wsj.com/articles/court-says-oracle-software-code-entitled-to-copyright-protection-1399652818, 2014. Retrieved May 27 2014.

[7] P. Kirn. Lilypond: Free, beautiful music notation engraving for anyone. http://createdigitalmusic.com/2010/05/14/lilypond-free-beautiful-music-notation-engraving-for-anyone, 2014. Retrieved May 27 2014.

[8] K. Knopper. Adriane — audio desktop reference implementation and networking environment. http://www.knopper.net/knoppix-adriane/index-en.html, 2014. Retrieved May 27 2014.

[9] R. Koch. *The 80/20 Principal*. NICHOLAS BREALEY PUBLISHING, LONDON, 1998.

[10] N. Koneri. Copy your linux install to a different partition or drive. http://www.linuxjournal.com/content/copy-your-linux-install-different-partition-or-drive, 2009. Retrieved May 27 2014.

[11] L. Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, 2 edition, 1994.

[12] R. Lerner. Switching to chrom(ium). http://www.linuxjournal.com/content/switching-

chromium, 2013. Retrieved May 27 2014.

[13] R. Love. Get on the d-bus. http://www.linuxjournal.com/article/7744, 2005. Retrieved May 27 2014.

[14] C. Newham. *Learning the bash Shell*. O'Reilly Media, 3 edition, 2005.

[15] Oracle corp. Java hotspotâĎć virtual machine performance enhancements. http://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html, 2014. Retrieved May 27 2014.

[16] L. Poettering. systemd, two years later. https://archive.fosdem.org/2013/interviews/2013-lennart-poettering/, 2013. Retrieved May 27 2014.

[17] M. Pozhidaev. Speech server voiceman. http://marigostra.com/projects/voiceman/, 2013. Retrieved May 27 2014.

[18] M. Pozhidaev. The text-based environment for blind persons: conception and operating system design. *Mezhdunarodnyj nauchno-issledovatelskij zhurnal*, 2:63–66, Feb. 2013.

[19] T.V Raman. Chromevox: built-in spoken feedback for chrome os. http://googlecode.blogspot.com/2011/05/chromevox-built-in-spoken-feedback-for.html, 2011. Retrieved May 27 2014.

[20] T.V Raman. Emacspeak — the complete audio desktop. http://emacspeak.sourceforge.net/, 2014. Retrieved May 27 2014.

[21] L. Shiman. X.org foundation releases x window system x11r6.7. http://lwn.net/Articles/79302/, 2004. Retrieved May 27 2014.

[22] R. Stallman. Gnu emacs manual. http://www.gnu.org/software/emacs/manual/pdf/emacs.pdf, 2013. Retrieved May 27 2014.

[23] J. Tranter. Dialog: An introductory tutorial. http://www.linuxjournal.com/article/2807, Sept. 1994. Retrieved May 27 2014.

[24] J. W. Valvano. *Embedded Systems: Introduction to ARM Cortex-M Microcontrollers cover*. CreateSpace Independent Publishing Platform, 2012.

[25] R. Yuen. Introducing networkmanager. http://www.redhat.com/magazine/003jan05/features/networkmanager/, 2005. Retrieved May 27 2014.

[26] D. Zeuthen. Simpler, faster, better. http://davidz25.blogspot.com/2012/03/simpler-faster-better.html, 2012. Retrieved May 27 2014.