Linux Audio Usability Issues

Introducing usability ideas based on Linux audio software

Christoph Eckert
Graf-Rhena-Straße 2
76137 Karlsruhe, Germany
mchristoph.eckert@t-online.de
February 2005

Abstract

The pool of audio software for Linux based operating systems has to offer very powerful tools to grant even average computer users the joy of making music in conjunction with free software. However, there are still some usability issues. Basic usability principles will be introduced, while examples will substantiate where Linux audio applications could benefit from usability ideas.

Keywords

Usability & application development

1. Introduction

Free audio software has become very powerful during the last years. It is now rich in features and mature enough to be used by hobbyists as well as by professionals.

In the real world, day by day we encounter some odds and ends which are able to make us unhappy or frustrated. May it be a screw driver which does not fit a screw, a door which does not close properly or a knife which simply is not sharp enough.

This is also valid for software usage. There often are lots of wee small things which sum up and prevent us from doing what we originally wanted to do. Some of these circumstances can be avoided by applying already existing usability rules to Linux audio software.

Usability usually is associated with graphical user interface design, mainly on Mac OS or even Microsoft Windows operating systems. Surprisingly, most of the rules apply to any software on any operating system, including command line interfaces.

A bunch of documents discovering this area of programming are available from various projects and companies, e.g. the GNU project, the Gnome desktop environment, the wxWidgets project[1], Apple computers or individuals.

One of the basic questions is how much a user needs to know before he is able to use a tool in order to perform a certain task. The amount of required knowledge can be reduced by clever application design which grants an average user an immediate start. Last but not least, clever software design reduces the amount of documentation needed; developers dislike writing documentation as well as users dislike reading it.

2. Usability Terms

Besides the classical paper »Mac OS 8 Human Interface Guidelines« by Apple Computer, Inc[2], the Gnome project has published an excellent paper discovering usability ideas[3].

Joel Spolsky has written a book called »User Interface Design for Programmers«. It is based on Mac and Windows development, but most ideas are also valid for Linux. Reduced in size, it is also available online[4].

To get familiar with the topic, some of the commonly found usability terms will be introduced. Included examples will concretize the ideas. Most of the examples do not only belong to one usability principle but even more.

2.1 Knowing the Audience

In order to write software which enables a user to perform a certain task, it is necessary to know the audience. It is a difference if the software system will teach children to read notes or enable a musician to create music. Different groups use a different vocabulary, want to perform different tasks and may have different computer knowledge.

Each user may have individual expectations on how the software will work. The expectations are derived from the task he wants to perform. The user has a model in mind, and the better the software model fits the user model, the more the user will benefit. This can be achieved by creating use cases, virtual users or asking users for feedback, so some applications include feedback agents.

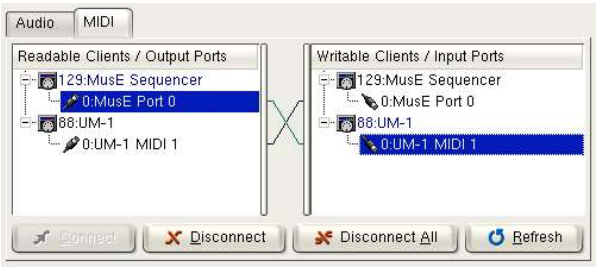
To fit the user's expectations is one of the most important and most difficult things. If the same question appears again and again in the user mailing lists, or even has been manifested in a list of frequently asked questions (known as FAQ), it is most likely that the software model does not fit the user model.

The target group of audio applications are musicians. They vary in computer skills, the music and instruments they play and finally the tasks they want to perform using the computer. Some want to produce electronic music, others want to do sequencing, hard disk recording or prepare scores for professional printing.

An audio application of course uses terms found in the world of musicians. On the other hand too specialized terms confuse the user. A piano teacher, for example, who gets interested in software sequencers, gets an easier start if the software uses terms he knows. A tooltip that reads »Insert damper pedal« can be understood more easily than »Create Controller 64 event«.

As soon as a user starts an audio software, he might expect that the software is able to output immediately sound to the soundcard. As we are on Linux, however, the user first needs to know how to set the MIDI and maybe the JACK connections.

An application therefore could make this easily accessible or at least remember the settings persistently until the next start and try to reconnect automatically. MusE for example is already doing so:



2.2 Metaphors

Metaphors are widely used to make working on computers more intuitive to the user, especially (but not only) in applications with graphical user interfaces (also known as GUI applications). Metaphors directly depend on the audience, because they often match things the user knows from the real world.

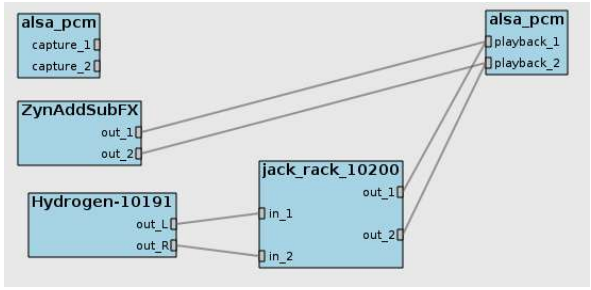
Instead of saving an URL, a bookmark gets created while surfing the web. On the other hand, choosing bad metaphors is worse than using no metaphor at all.

Well chosen metaphors reduce the need to understand the underlying system and therefore ensure that the user gets an immediate start. In the following example, a new user will be most probably confused by the port names:

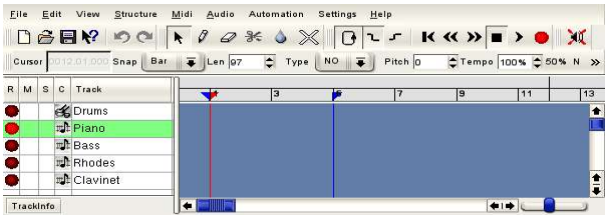
```
bash-2.05b$ jack_lsp
alsa_pcm:capture_1
alsa_pcm:capture_2
alsa_pcm:playback_1
alsa_pcm:playback_2
alsa_pcm:playback_3
alsa_pcm:playback_4
alsa_pcm:playback_5
alsa_pcm:playback_6
bash-2.05b$
```

A metaphor makes it easier to the user to understand what is meant, maybe Soundcard input 1 and 2 and Soundcard output 1 through 6 instead of alsa_pcm:capture or alsa_pcm:playback.

Patchage[6] is a nice attempt to use metaphors to visualize the data flow in a manner the audio user is familiar with, compared to connecting real world musical devices. It could become the right tool for intuitively making MIDI connections as well as JACK audio connections. If it contained a LADSPA and DSSI host, it could be a nice tool to easily control most aspects of a Linux audio environment:



An example for a misleading metaphor is an LED emulation used as a switch. An LED in real life displays a status or activity. A user will not understand that it is a control to switch something on and off. In MusE, LEDs are used to en- and disable tracks to be recorded, and this often is not clearly understood by users:



Replacing the LEDs by buttons which include an LED to clearly visualize the recording status of each track would make it easier to understand for the user.

2.3 Accessibility

All over the world, there are citizens with physical and cognitive limitations. There are blind people as well as people with hearing impairments or limited movement abilities. On a Linux system it is easily possible to design software which can be controlled using a command line interface. When designing GUI software it is still needed to include the most important options as command line options. This way even blind users are able to use a GUI software synthesizer by starting it with a certain patch file, connecting it to the MIDI input and playing it using an external keyboard controller.

Free software often gets used all over the world. It is desirable that software is prepared to easily get translated and localized for different regions. By including translation template files in the source tarball and the build process, users are able to contribute by doing translation work:

Besides Internationalization and Localization (both also known as i18n and l10n), accessibility includes paying attention to cultural and political aspects of software. Showing flags or parts of human beings causes unexpected results, maybe as soon as an icon with a hand of a western human will be seen by users in Central Africa.

Keyboard shortcuts enable access to users who have problems using a mouse. The Alt key is usually used to navigate menus, while often needed actions are directly accessible by Ctrl-keycombos. The tabulator key is used to cycle through the controls of dialogs etc.

2.4 Consistency

Consistency is divided into several aspects of an application. It includes consistency with (even earlier versions of) itself, with the windowmanager used, with the use of metaphors and consistency with the user model.

Most of the time, users do not only use one application to realize an idea. Instead, many applications are needed to perform a job, maybe to create a new song. The user tries to reapply knowledge about one application while using another. This also includes to make an application consistent with other applications even if something has been designed wrong in other applications. If an application is a Gnome application, the user expects the file open dialog to behave the same as in other Gnome applications. Writing a new file request dialog in one of the applications will certainly confuse the user even if it was better than the generic Gnome one.

Consistency does not only affect GUI programs but command line programs as well. Some Linux audio programs can be started with an option to use JACK for audio output using a command line parameter. As soon as applications behave differently, the user cannot transfer knowledge about one program to another one:

```
bash-2.05b$ aeolus -J
bash-2.05b$ ams --jack
bash-2.05b$ mplayer -ao jack
```

There are also programs which do not read any given command line parameters. Invoking such a program with the help parameter will simply cause it to start instead of printing some useful help on the screen:

```
bash-2.05b$ ratonx --help
Start: multinotes_on
trigger: mouse button pressed
mouse button: 1
datatype: data statement
Destination: ALSA seq note on
data: (12), index: 0(50), index: 1(80), index:
 2
```

The GNU coding standards[5] recommend to let a program at least understand certain options. To ask a program for version and help information should really included in any program. Even if there is no help available, it is helpful to clearly state this and point the user to a README file, the configuration file or an URL where he can get more information. If a GUI application contains a help menu, it is useful if it at least contains one entry. It is better to have a single help page clearly stating that there is no help at all and pointing to the project homepage or to a README file than having no help files installed at all.

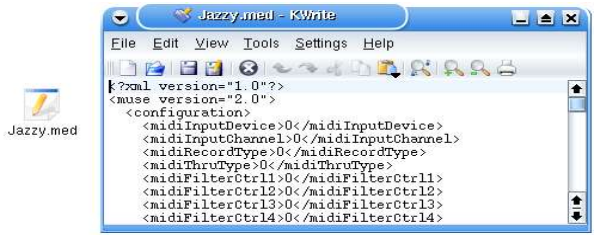
Programs containing uppercase characters in the name of the binary file confuse the user. The binary file of KhdRecord for example really reads as »KhdRecord« making it difficult for the user to start it from a command line, even if he remembers the name correctly. Another example are program's where the binary file name does not fit the application's name exactly, as found on the virtual

keyboard. The user has to guess the binary representation, and this causes typing errors:

```
bash-2.05b$ vkeyboard
bash: vkeyboard: command not found
bash-2.05b$ vkboard
bash: vkboard: command not found
bash-2.05b$ virtualkbd
bash: virtualkbd: command not found
bash-2.05b$ vkeybrd
bash: vkeybrd: command not found
bash-2.05b$ vkeybd
```

GUI programs can add a menu entry to the system menu so the user is able to start the program the same way as other programs and doesn't need to remember the application's name. Therefore, GUI applications must not depend on command line options to be passed and need to print important error messages not only on the command line, but also via the graphical user interface.
For consistency reasons, such desktop integration shouldn't be left to the packagers. The user should always find an application on the same place in the menu system, regardless which distribution he is running.

Users are used to click on documents to open these in the corresponding application. So it is useful if an audio application registers the used filetypes. MusE for example saves its files as *.med files. Due to the lack of filetype registration, KDE recognizes these files as text files. Clicking on »Jazzy.med« will open it in a text editor instead of starting MusE and loading it:



Consistency also includes the stability of an application, security aspects and its data compatibility with itself as well as other applications. Even in the world of free software it is often not simple for the user to exchange data between different applications.

2.5 Feedback

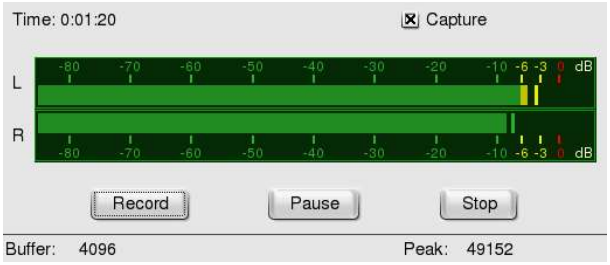
A computer is a tool to enter, process and output data. If a user has initiated a certain task, the system should keep him informed as long as there is work in progress or if something went wrong. This way the user doesn't have to guess the status of the system. The simpler a notification is, the better the user will be able to understand and to remember it after it is gone.

Providing information which enables the user to solve a problem and to avoid it in the future reduces disappointment and frustration. For the same reason message texts should be designed in a manner that the software instead of the user is responsible for errors.

When starting MusE without a JACK soundserver already running, the user gets prompted by an alert clearly explaining the problem:

The user now has a good starting point and learns how to avoid this error in the future.

A further example for good feedback is found in qarecord[7]. It clearly shows the input gain using a nice level meter the user may know from real recording equipment:



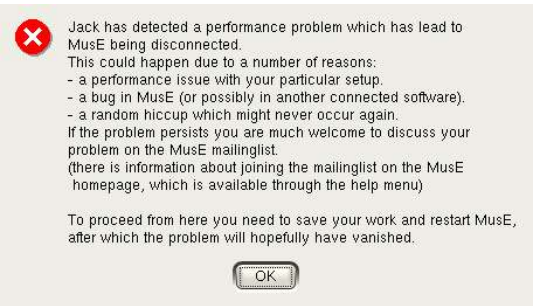
On the other hand, the user gets no feedback if currently audio is recorded or not. If no recording is in progress it makes no sense to press the pause and stop buttons, so both need to appear inactive. As soon as recording is in progress it makes no sense to press the record button again, so the record button needs to be set inactive. If the recording is paused, the pause or record button needs to be disabled.
Qarecord needs to be started in conjunction with some command line parameters defining the hardware which should be used to capture audio. If qarecord included a graphical soundcard and an input source selector as well as a graphical gain control, it would perfectly fulfill further usability ideas like direct access and user control. The user was able to do all settings directly in qarecord instead of using different applications for a simple job.

To offer feedback, it is necessary to add different checks to a software system. A user starting an audio application might expect it will immediately be able to output sound. On Linux based systems, there are some circumstances which prevent an application from outputting audio. This for example happens as soon as one application or soundserver blocks the audio device while another one also needs to access it. In this case, there are applications which simply seem to hang or even do not appear on the screen instead of giving feedback to the user. Actually, the application simply waits until the audio device gets freed.
In the following example, xmms gets started to play an audio file. After that, Alsa Modular Synth (AMS) gets started, also directly using the hardware device, while xmms still is blocking the device. Unfortunately, AMS does not give any feedback, neither on the command line nor in the graphical user interface:

```
bash-2.05b$ xmms Tower_Of_Integration.mp3 &
[1] 9375
bash-2.05b$ ams --soundcard hw:0,0
LADSPA_PATH: /usr/lib/ladspa
loadPath: /usr/share/doc/, savePath: /home/ce/
```

The user will only notice that AMS does not start. As soon xmms will be quit, even if it were hours later, AMS will suddenly appear on the screen. Some feedback on the command line and a graphical alert message helped the user to understand, to solve and to avoid this situation in the future.

Concerning JACK clients, it is an integral part of the philosophy that applications can be disconnected by jackd. As soon as this happens, some applications simply freeze, need to be killed and restarted.
MusE shows an informational dialog instead of simply freezing:



Of course it would be better if jackified applications would not need to be restarted and could try to automatically reconnect to JACK without any user interaction as soon as a disconnect occures.

2.6 Straightforwardness

To perform a task, the user has to keep several things in mind, may it be a melody, a drum pattern etc. There are external influences like a ringing telephone or colleagues needing some attention. So, the user only spends a reduced amount of his attention to the computer. This is why applications need to behave as straightforward as possible.

One of the goals is that the computer remembers as many things as possible, including commands, settings etc. On Linux, the advantages of the command line interface are well known, but it is also known how badly commands and options are remembered if these are not used or needed often. This is why typed commands are replaced by menus and buttons in end user software whenever possible.

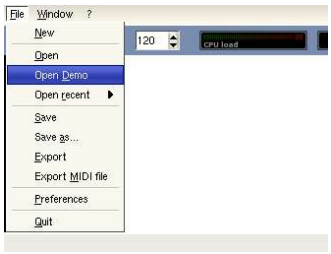
Users dislike reading manuals or dialog texts and developers dislike writing documentation. Keeping the interface and dialogs straightforward will reduce the need of documentation. It is also true that it is difficult to keep documentation in sync with the software releases.

It is also important to create a tidy user interface by aligning objects properly, grouping them together and giving equal objects the same size.

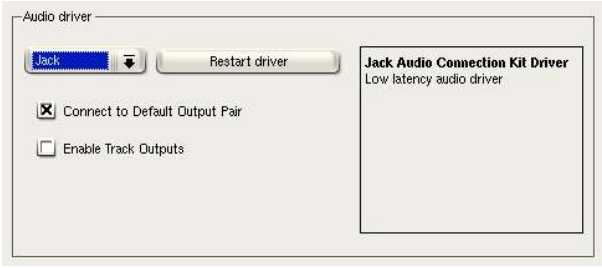
Disabling elements which do not fit the current situation helps the user to find the right tools at the right time. Ordering UI elements to fit the user's workflow reduces to write or to read documentation. Think about analogue synthesizers: Mostly, the elements are ordered to fit the signal flow from the oscillators through the mixer and filter sections to the outputs.

According to Dave Phillips, who asked the developers to write documentation, an additional thing is to reduce the amount of documentation needed. The best documentation is the one which does not need to be written and the best dialogs are the ones which do not need to appear. Both will reduce the time of project members spent on writing documentation and designing dialogs. The user will benefit as well as the project members.

After invoking an application, the user likes to start immediately to concentrate on the task to perform. He might expect a reasonable preconfigured application he immediately can start to use. When starting, Hydrogen opens a default template file. Further demo files are included, easily accessible via the file menu:

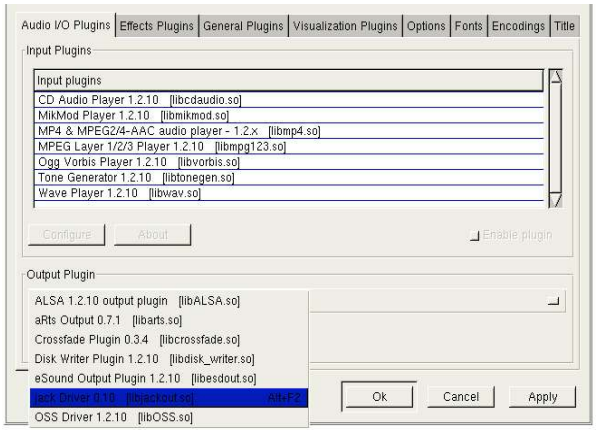


Hydrogen optionally remembers the last opened file to restore it during the next start and automatically reconnects to the last used JACK ports:



Alsa Modular Synth includes a lot of example files, too, but it does not load a template file when starting, and there is no easy access to the example files. The user needs to know that there are example files, and the user needs to know where these files are stored.

Another example is the fact that there are different audio subsystems on a Linux box. An audio application which wants to work straightforward included various output plugins. One application which already has a surprising amount of output plugins (including a JACK plugin) is xmms:



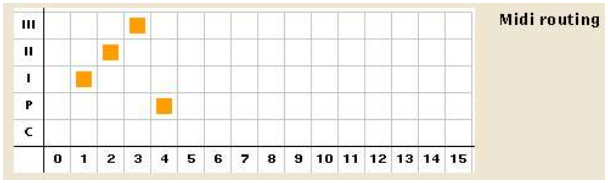
Furthermore, xmms offers feedback as soon as the configured output destination is not available during startup:

A further idea to make it even more straightforward could be that xmms did some checks as soon as the preconfigured device is not available during startup and chooses an alternative plugin automatically. If doing these checks in an intelligent order (maybe JACK, esound, aRts, DMIX, ALSA, OSS), xmms will most probably match the user's expectations. If no audio system was found, an error message got printed.

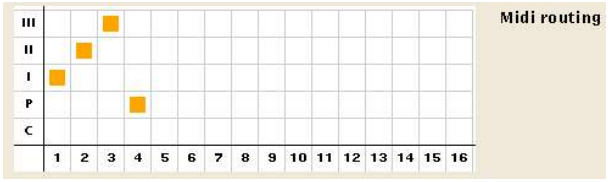
Introducing such a behaviour could improve other Linux audio applications, too. Some example code in the ALSA Wiki pages[8] could be a good starting point for future audio developers.

Average human beings tend to start counting at 1. In the software world, counting sometimes starts at zero for technical reasons. Software makes it possible to hide this in the user interface. This includes the numbering of sound cards (1, 2, 3 instead of 0, 1, 2) as well as program changes (1, 2, 3 ... 128 instead of 0, 1, 2 ... 127) or MIDI channels (1, 2, 3 ... 16 instead of 0, 1, 2 ... 15).

A musician configured his keyboards to send notes on the channels 1 through 4. If the software starts the numbering at zero, the user has to struggle with the setup:



A more human readable numbering matched the user's expectations much more:



2.7 User Control

Every human wants to control his environment. Using software, this includes the ability to cancel long lasting operations as well as the possibility to configure the system, like preferred working

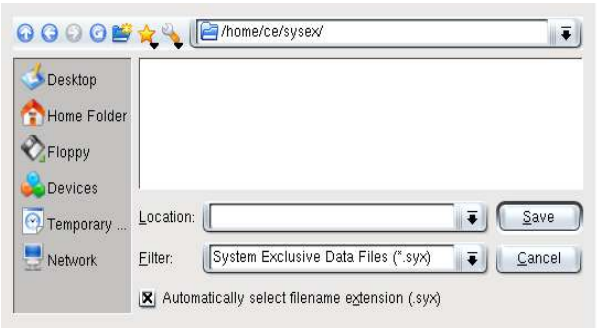
directories and the preferred audio subsystem. An application that behaves the way the user expects makes him feel that he is controlling the environment. It also needs to balance contrary things, allowing the user to configure the system while preventing him from doing risky things.

As soon as a musician gets no audio in a real studio, he starts searching where the audio stream is broken. In a software based system, he also needs controls to do so. Such controls are not only needed for audio but also for MIDI connections. An LED in a MIDI sequencer indicating incoming data or a levelmeter in an audio recording program to indicate audio input are very helpful to debug a setup. On some hardware synthesizers corresponding controls exist. A Waldorf Microwave, for example, uses an LED to indicate that it is receiving MIDI data. Rosegarden displays a small bar clearly showing that there is incoming MIDI data on the corresponding track:

A further thing worth some attention is not to urge the user to make decisions. SysExxer is a small application to send and receive MIDI system exclusive data (also known as sysex). After SysExxer has received sysex data the user must decide that the transmission is finished by clicking the OK button:

After that the user has to decide if the data gets saved as one single or as multiple split files:

SysExxer then asks for a location to store the file:



SysExxer urged the user to make three consecutive decisions and therefore the user does not feel like he is controlling the situation. Instead, the application controls the user.

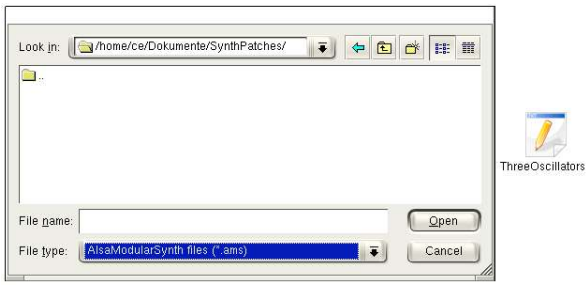
Some Qt based audio programs often forget the last used document path. The user opens a file somewhere on the disk. As soon he wants to open another one, the application has forgotten the path the user has used just before and jumps back to the home directory:

Most probably the user did expect to be put back to the last used directory. Applications can offer preferences for file open and file save paths or even better persistently remember the last used paths for the user.

The Linux operating system does not depend on file extensions. On the other hand, file extensions are widely used to assign a filetype to applications. If an application has chosen to use a certain file extension, it should ensure that the extension gets appended to saved files even if the user forgot to specify it.

A file open dialog can filter the directory contents so the user only gets prompted with files matching the application. On the other hand, a file open dialog also should make it possible to switch this filter off, so the user is able to open a file even if it is missing the correct extension.

If an application does not ensure that the extension gets appended when saving a file and does not enable the user to switch the filter off when trying to reopen the file, it will not appear in the file open dialog even if it resides in the chosen directory, so the user is unable to open the desired file:



On Linux configuration options are usually stored in configuration files. A user normally doesn't care about configuration files because settings are done using GUI elements. On the other hand, sometimes things go wrong. Maybe a crashing application will trash its own configuration file or an update has problems reading the old one. Therefore, command line options to ask for the currently used configuration and data files are sometimes very helpful. The same information can be displayed on a tab in the about box. This enables the user to modify it if needed.

User control includes to enable the user to configure the base audio environment. Maybe he wants to rename soundcards and ports according to more realistic device names like »Onboard Soundcard Mic In« or similar. If a MIDI device which offers multiple MIDI ports is connected to the computer, it is useful to be able to name the ports according to the instruments connected to the ports. If there is more than one soundcard connected, the user may like to reorder them in a certain manner, perhaps to make the new USB 5.1 card the default device instead of the onboard soundchip. He wants to grant more than one application access to a soundcard, in order to listen to ogg files while a software telephone still is able to ring the bell.

In the last few years, alsaconf has done a really great job, but meanwhile it seems to be a little bit outdated because it is unable to configure more than one soundcard or to read an existing configuration. It is unable to configure the DMIX plugin or to put the cards in a certain user defined order. It still configures ISA but no USB cards.

A replacement seems to be a very desirable thing. Such a script, used by configuration front ends, will bring many of the existing but mostly unused features to more users.

A further script could be created to help configuring JACK, maybe by doing some tests based on the hardware used and automatically creating a base JACK configuration file.

2.8 Forgiveness

A software system which allows to easily undo more than one of the last actions performed will encourage the user to explore it. The classical undo and redo commands are even found on hardware synthesizers like the Access Virus[9].

Some applications allow to take snapshots so the user can easily restore a previous state of the system, for example on audio mixing consoles.

As soon the user wants to do an irreversible action an alert should ask for confirmation.

2.9 Direct Access

In graphical user interfaces tasks are performed by manipulating graphically represented objects. It is a usability design goal to make accessing the needed commands as easy as possible.

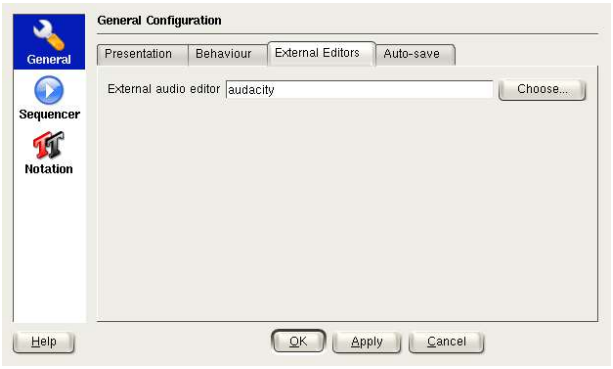
Options an application supports should be made dynamically accessible, even during runtime. AMS for example supports a monophonic as well as a polyphonic mode via a command line option. Unfortunately, it is not possible to enter the desired polyphony persistently via the preferences or to change the polyphony during runtime. As soon as the user forgets to pass the desired polyphony to AMS during startup, it needs to be quit and restarted with the correct polyphony applied. Then the user has to reopen the patch file and to redo all MIDI and audio connections. Therefore there is no direct access to the polyphony settings.

When a user changes the preference settings of an application, the program should change its behaviour immediately. It is also important to write the preferences file immediately after changes have been made. Otherwise, a crash will make the application forget the settings the user has made. In gAlan for example, preference changes make it necessary to restart the program:

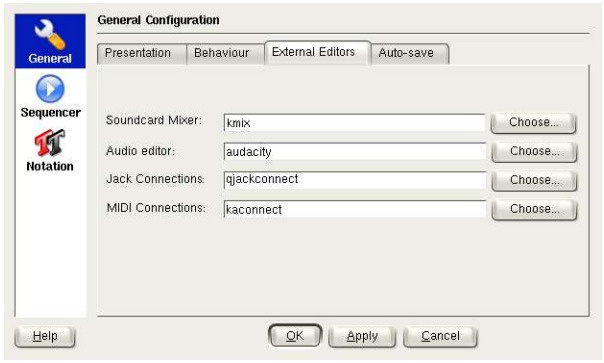
Tooltips and »What's this« help are very useful things in GUI applications. Both grant the user direct access to documentation on demand and reduce the need for writing documentation. On the other hand, if an application offers tooltips and

»What's this« help, both need to contain reasonable contents. Otherwise, the user may believe that similar controls will also be useless in other applications.

A further thing to grant users direct access is to give them needed options and controls where and when needed, even if the controls were external applications. A Linux audio user often is in need to access the soundcard's mixer as well as a MIDI or JACK connection tool. Therefore, a sequencer application offering a button to launch these tools from within its user interface grants the user direct access. There are different tools like kaconnect, qjackconnect or qjackctl, so preference settings make it possible that the user enters the programs he likes to use. Rosegarden for example allows the user to enter the preferred audio editor:



The same way external MIDI and JACK connection tools as well as a mixer could be made available:



These fields need to be well preconfigured during the first application startup. One possibility is to set the application's defaults to reasonable values during development time. Unfortunately, this seems to be impossible because the configuration of the user's system is not known at this moment.

Therefore, Rosegarden could try to check for the existence of some well known tools like qjackctl, qjackconnect, kaconnect, kmix, kamix or qamix and enter them into the matching fields in case these are empty at application startup.

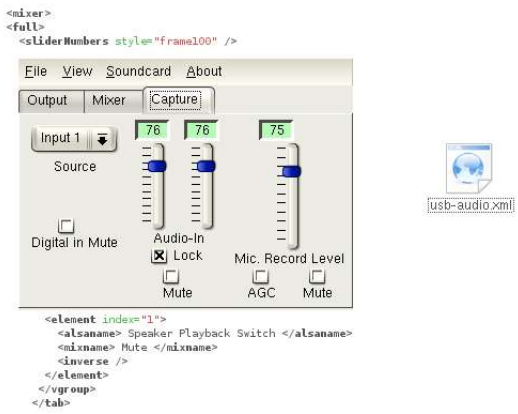
An application could even offer the possibility to make MIDI and audio connections directly from its interface with UI elements of its own. As long as the connections reappear in aconnect and jack_connect, it fulfills both the usability requirements of direct access and consistence.

A further issue concerning direct access is setting a soundcard's mixing controls. On a notebook with an AC '97 chip, alsamixer usually shows all controls the chip has to offer, regardless if all abilities of the chip are accessible from the outside of the computer or not:

Of course ALSA cannot know which pins of the chip are connected to the chassis of the computer. The snd-usb-module really handles a huge amount of more and more hotpluggable devices. This simply makes it impossible for ALSA to offer a reasonable interface for each particular card. Currently, alsamixer for a Terratec Aureon 5.1 USB looks like this:

PCM1 does not seem to do anything useful, while auto gain is not a switch (like the user might expect) but looks like a fader instead. It cannot be faded, of course, but muted or unmuted to switch it on and off. There are two controls called 'Speaker'. One of them controls the card's direct throughput, while the second one controls the main volume. The average user has no chance to understand this mixer until he plays around with the controls and checks what results are caused by different settings.

Qamix tries to solve this building the UI reading an XML file matching the card's name:

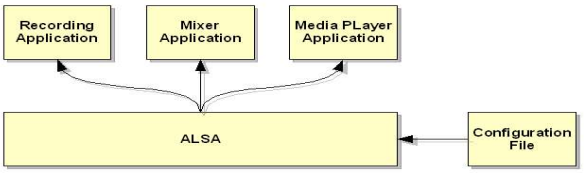


The user configures a card by adjusting this file. This is a nice attempt, but still qamix gets too less information from ALSA to make a really good UI:

```
bash-2.05b# qamix -p
<mixer>
<full>
      <element>
        <alsaname> Master Mono Playback Switch </alsaname>
        <mixname> Master Mono Playback Switch </mixname>
      </element>
      <element>
        <alsaname> Master Mono Playback Volume </alsaname>
        <mixname> Master Mono Playback Volume </mixname>
      </element>
```

A further usability goal is consistency. All these points mentioned above require that ALSA needs to remain the central and master instance for all user applications.

One idea to realize this is to make ALSA able to give applications more and better information about the hardware. As soon a soundcard gets configured, a matching, human created configuration file got selected manually or via a card ID, so the controls of a particular card could be named more human readable. If the configuration file was in Unicode format, it even can contain names in different languages:



Introducing a similar system for ALSA device and MIDI port names also seems to be desirable. This needs writing some configuration files. Keeping the file format as simple unicode text or XML files which simply are stored at a certain loaction can make users easily contribute.

3. The Developer's Point of View

After having discussed a lot of usability issues from a user's point of view, it is also necessary to view it from a developer's point of view. There are several reasons why it is difficult to introduce usability into Linux audio software projects.

First of all, the question is if there is any reason why a developer of free software should respect usability ideas. If someone has written an application for his own needs and has given it away as free software, the motivation to apply usability ideas has a minor priority. Nobody demands that an application that someone submitted for free has to include anything.

In the beginning of a software project, it makes less sense to think about the user interface. At first, the program needs to include some useful functionality. As soon as a project grows and maybe several developers are working on it, the user base might increase. In this case, the ambition of the project members to make the application better and more user friendly usually increases.

Even in this case, it is often difficult to achieve a more user friendly design for technical reasons. Applications often are designed to start with command line parameters to make them behave as desired. If the design of the software is made to take options at application startup, it is likely that these cannot easily be changed during runtime. The same is valid for preference settings read from a configuration file during an application's startup. It often does not expect the values to be changed dynamically. Adjusting this behaviour at a later point in the development process is sometimes difficult and can cause consecutive bugs.

Keeping this in mind, it requires to design the software system to keep the different values as variable as possible from the very beginning of the development process. Changing this afterwards can be difficult.

A further point is the fact that developers tend to be technically interested. A developer might have been working on new features for a long time, having done research on natural phenomena and having created some well designed algorithms to simulate the result of the research in software. The developer now is proud having created a great piece of software and simply lacks the time to make this valuable work more accessible to the users. The developer has put a lot of time and knowledge in the software and then unfortunately limits the user base by saving time while creating the user interface.

Sometimes it is also caused by the lack of interest about user interface design. There have always been people who are more interested in backends and others more interested in frontends. Developers who are interested in both cannot be found often. It is important to notice that this is a

given fact. But not each project has the luck that one of the members is an usability expert, so it is useful if all project members keep usability aspects in mind.

Of course, bugs need to be fixed, and even the existing user base tends to beg more for new features instead of asking to improve the existing user interface. This is understandable because the existing user base already knows how to use the program.

A software system will never be finished. So developers who want to introduce usability improvements need to clearly decide if they want to spent some time on it at a certain point.

Working on usability improvements needs time, and time is a strongly limited resource especially in free software projects which mainly are based on the work of volunteers.

Everything depends on the project members and if they are interested in spending some time on usability issues or not.

4. Summary

Linux is surely ready for the audio desktop. Most applications a musician needs are available, at least including the base functionality. Furthermore, there is free audio software which does things musicians have never heard about on other operating systems.

Due to this fact, keeping some usability ideas in mind will make more semi-skilled users enjoying free software. More users cause more acceptance of free software, and this will cause more people to participate.

Usability is not limited to graphical user interfaces. It also affects command line interfaces. Linux is known to be a properly designed operating system. Respecting some basic usability rules helps continuing the tradition.

On Linux there are many choices which environment to work in. There are different window managers and desktop environments as well as different toolkits for graphical user interface design.

Working and developing on a heterogeneous system like Linux does not mean that it is impossible or useless to apply usability rules. It simply means that it is a special challenge to address and work on usability ideas.

Paying attention to usability issues is not only important to make user's life easier or improve his impression. It also is important to broaden the user base in order to get more bug reports and project members. And finally it helps spreading Linux when surprising average computer skilled musicians what cool applications are available as free software.

5. License

6. Resources

- [1] The wxWidgets toolkit wxGuide: http://wxguide.sourceforge.net
- [2] The Apple human interface principles: http://developer.apple.com/documentation/mac/pdf/HIGuidelines.pdf
- [3] The Gnome user interface guidelines: http://developer.gnome.org/projects/gup/hig/
- [4] User Interface Design by Joel Spolsky: http://joelonsoftware.com /navLinks/fog0000000247.html
- [5] The GNU coding standards: http://www.gnu.org/prep/standards
- [6] Patchage: http://www.scs.carleton.ca/~drobilla/patchage/
- [7] Various software of M. Nagorni: http://alsamodular.sourceforge.net/
- [8] The ALSA wiki pages: http://alsa.opensrc.org
- [9] The Access Virus Synthesizers: http://www.virus.info/