Accessibility meets Usability Weekend

Minutes from a Meeting
of Members of the

Linaccess-Team,
KDE Accessibility and Usability,
and OpenUsability
in February 2006

In Cooperation With
barrierefrei kommunizieren!
Berlin

May 2006

Table of Contents

Accessibility meets Usability Weekend

Minutes from a Meeting of members of the

linaccess-team,

KDE accessibility and Usability,

and OpenUsability

0. Introduction

Making a software accessible for handicapped users is more and more becoming a sales argument - not only for proprietary software, but also for Open Source solutions as the current discussions about applications that support the Open Document Format show [1].

On the user side, there is as well a high demand for accessible Open Source solutions: Proprietary software supporting the requirements of certain handicapped user groups is highly specialised software, and therefore very expensive. Making FLOSS solutions accessible would be a huge benefit for many communities of handicapped computer users all around the world.

There are several efforts that try to accomplish that goal - some coming from the major Linux desktop environments themselves like the Gnome and KDE accessibility team, others, as the linaccess group, are independent. In most of these teams, technical questions are in the center of attention. The usability of most accessibility features has so far been second range in these efforts.

Therefore, Lars Stetten from the Linaccess Team and a group of KDE accessibility and usability people decided to test the usability of accessibility features. Five members of the linaccess team, three being partially sighted and two blind, volunteered to perform monitored usability tests.

As a subject of research, several KDE features for partially sighted people and the Gnopernicus screen reader for Gnome were chosen. The goal of the usability tests was not to achieve statistical data, but to gain an understanding of the needs of the represented user types.

As a general conclusion of the test, it was found that while both KDE and Gnome provide very good tools to make the Linux desktop usable for partially sighted and blind users, they are lacking consistent support among the major desktop applications. In KDE, key applications like the text editor Kate or the shell Konsole did not apply high contrast colour schemes; in Gnome (Ubuntu), the contents of crucial tools like the software installation could not be read by Gnopernicus and were therefore “invisible” for the blind users. Alongside this, a number of insights could be won on how to design a feasible document reader for KDE, how to improve magnifiers, and what elements in Gnome applications require Gnopernicus integration.

1. Methods

1.1 Objectives and Procedure

To improve the usability of accessibility features in Open Source software, usability tests concentrating on the support for partially sighted and blind users were performed with the two major Linux desktops KDE and Gnome. The goal was not to achieve statistical data, but to gain an understanding of the needs of the represented user types.

Regarding partially sighted users, colour schemes, screen magnifiers and document readers in KDE were subject of the testings. Regarding blind users, the integration of the screen reader Gnopernicus with Ubuntu was probed.

Other planned tests, such as a Kubuntu installation script, the new accessibility options of the Ubuntu installation CD or the IBM screen magnifier for Linux had to be skipped due to time restrictions.

Most of the testing was performed in single sessions with two or more observers, only the initial Gnopernicus "warm-up" session was simultaneously accomplished with the two blind participants. The sessions were a combination of explorative usability testing, unstructured interviews, and task observation. That means users were given tasks they should perform with a certain tool or desktop configuration, but at the same time were thoroughly interviewed about reasons for their behaviour, habits and preferences regarding their home system, and sometimes stepped back from the computer to explain in detail why certain options need to be one way and not the other. After the test, two participants showed how they usually worked in their common home environment. By this combination of different information sources, the moderators and observers were able to identify reasons for usage patterns, and could discuss possible solutions to problems contemporary to the testing.

1.2 Subject of Research and Use Scenarios

1.2.1 Support for Partially Sighted Users in KDE

Regarding partially sighted users, the KDE accessibility features were tested. Of special interest were the usability and integration of high contrast colour schemes with the KDE desktop, a new screen magnifier for KDE, and KDE document were tested. In Detail, the following tools and use scenarios were tested:

High contrast colour schemes:

There are several high contrast colour schemes available in KDE. In the tests, the "High Contrast Dark big" theme was tested. The users performed the following tasks:

- Launching Konqueror for web browsing (KMenu or Run command).
- Going to the website www.openusability.org using the location bar (Konqueror).
- Downloading a PDF which is linked in a news on OpenUsability.org (Konqueror).
- Changing the colors of the PDF to make it better readable for participant (KPDF).
- Type some text in a text editor (Kate).
- Change the colors to make text more readable for participants (Kate).

New screen magnifier for KDE:

The new screen magnifier for KDE came in different viewing modes, the most important ones being fisheye and fullscreen magnification. Furthermore, there were different invert algorithms which automatically exchanged colours displayed on the screen. The users were asked to perform the following tasks:

- Check your email and remove Spam (KMail).
- Write an email (Kmail).

KDE Document Readers:

In different KDE applications, the integration with the text-to-speech application Ktts is implemented in a different way: In some applications, it is invoked for marked text via the context-menu, in others via the menu. Other applications do not have a Ktts integration, so text needs to be copied and inserted into the Ktts main window. A fourth approach is KSayIt, an application that loads complete files and reads them in a separate application. The goal of testing these different approaches was to learn about benefits and problems of the different solutions as a base for the KDE4 integrated document reader. Regarding document readers, the users performed the following tasks:

- Have Konqueror read a web page aloud.
- Have KPDF read the PDF aloud.
- Start KSayIt and open a random document with it.
- Have KSayIt read a file aloud.

1.2.2 Support for Blind Users in Gnome

Regarding blind users, the Gnome screen reader Gnopernicus was tested. Of

special interest was the ease of installation, the integration with the major Gnome applications, as well as the general performance. In Detail, the following tools and use scenarios were tested:

Installation of the base system and of accessibility features in Ubuntu / Gnome:

The installation was performed by two blind users and two partially sighted users, each experienced with Linux system administration. The installation covered the following areas:

- todo: Please check!
- Ubuntu base installation.
- Gnopernicus installation.
- Braille-support.

Gnopernicus integration in Gnome:

When the installation was complete, the following tools and use cases were tested:

- Search a file on a USB stick and burn it on CD (Nautilus).
- Read a PDF (Acrobat Reader).
- Write some text in a text editor (GEdit).
- Install the email application Thunderbird (Add/Remove Software).
- Check your email and remove Spam (Evolution, Thunderbird).
- Set up your IM account and chat with somebody (Gaim).

Another planned task was internet browsing, but the four participants taking care of the installation did not succeed to set up German language packages in an appropriate way that would allow for browsing German web pages.

1.3 Participants

Five members of the linaccess team, three being partially sighted and two blind, volunteered to perform monitored usability tests.

1.3.1 Partially Sighted Users

Two partially sighted users, Lars Stetten and Christoph Niehaus, were in need of a dark colour scheme (black background, bright font) in order to avoid being dazzled. Both required a threefold zoom, corresponding to a font size of 22 to 28.

Christoph has been a Windows user since he started using computers. Since 2000, he has additionally installed a Linux system which he is using on a regular base, but Windows staying his major system due to an optimal screen magnifier.

On his windows system, Christoph makes use of the screen magnifier ZoomText and its colour invert algorithms. On Linux, he usually uses the high contrast colour schemes, high font sizes and an increased virtual resolution.

He has set up his work environment to perfectly fit his needs, for example a 21 inch monitor that was positioned in a distance of 15 centimeters before his eyes, and a multi-functional mouse whose buttons were tied to frequently used operations.

Lars has been a KDE user for five years, and could therefore demonstrate what parts of the system needed manual adjustment to fit his needs, and how he usually worked with a computer.

Lars makes use of multiple virtual desktops, each running one application in full screen mode. Usually he knows which application is running on which desktop, to navigate between the windows he therefore switches desktops via the keyboard. He seldom uses the taskbar or Alt-Tab. Regarding the panel, he mostly makes use of the system tray, and sometimes the KMenu or Konqueror.

Mirko Blinn, the third partially sighted user, has been a KDE user for one year. Other than Christoph and Lars he was not in need of inverted colour schemes. Also, his preferred zoom factor for the desktop was much lower: It lay around 1.25.

Mirko usually uses one virtual desktop running one application in full screen mode. When he needs to open more windows, he presses Alt-Tab to navigate between the windows. He therefore seldom needs to rely on the taskbar, and the panel mostly functions as a quick launcher for the KMenu or Konqueror.

1.3.2 Blind Users

Sebastian Andres and Henning Oschwald evaluated the current state of the Gnopernicus screen reader.

While Sebastian has been a user of the command line in combination with the screen reader sbl and brltty for several years, Henning was well familiar with Gnopernicus. At home, he used a combination of Gnopernicus for graphical user interfaces and sbl for the command line.

todo: more about henning and sebastian!

1.4 Technical Setup

The tests were conducted on different machines running different operating systems.

1.4.1 Colour Schemes

The tests on colour schemes in KDE 3.5 were performed on an Acer laptop with 15 inch screen, a resolution of 1024x768 and a small laptop mouse (nonstandard size), running Kubuntu 5.10 Breezy Badger.

Regarding the colour schemes, the following configuration was used:

- Colour scheme: "HighContrast White Text".
- Icons: "Monochrome" theme from the kdeaccessibility package (available on kde-look).
- Font and font size: 22 to 28.
- Kicker: Colour manually set to black.
- Clock applet: Colors manually set to black background and white text.
- Pager applet: Options manually set to "elegant" and "Desktop Wallpaper".
- Desktop Background: Colour manually set to dark blue.
- Konqueror: Webpage stylesheet manually set to "use accessibility stylesheet" in "white on black" mode.
- Mouse pointer: Colour manually changed to white, otherwise default settings as the pointer size could not be increased with Kubuntu.

Most of these settings are summarised in the “HighContrast Dark Big” KDE theme and can be chosen by “two clicks”.

Some more issues on colour schemes were evaluated on a desktop computer running Kubuntu 5.10 Breezy Badger provided by barrierefrei kommunizieren!, with a 21 inch screen which was run with a resolution of 1024x768. Otherwise, the equal accessibility settings were made as on the laptop.

Performing the tests on notebooks was suboptimal as the partially sighted participants had medium to severe problems to get used to the keyboard layout. Also, distance to the screen, screen size and reflection hindered them while performing the tasks. Connecting an external monitor did not work by technical reasons.

1.4.2 Screen Magnifier

The tests on screen magnifiers were performed on a notebook with a screen resolution of 1024x768, running SuSE Linux xx and KDE xx.

Performing the tests on notebooks was suboptimal as the partially sighted participants had medium to severe problems to get used to the keyboard layout. Also, distance to the screen, screen size and reflection hindered them while performing the tasks. Connecting an external monitor did not work by technical reasons.

1.4.3 Document Readers

The document readers were partly tested in combination with the colour schemes (hardware details see 1.4.1 Colour Schemes) the colour schemes, partly in combination with the screen magnifier (hardware details see 1.4.2 Screen Magnifier) .

1.4.4 Gnopernicus Screen Reader

The Gnopernicus tests were perfomed on a PC provided by barriefrei kommunizieren!. The participants installed Ubuntu 6.04 Dapper Drake - beta, as well as Gnopernicus version 1.0 and a Braille Input xxxx.

2. Results: Support for Partially Sighted Users in KDE

2.1 High Contrast Themes

High contrast themes were tested on two different computers, a notebook with a 15 inch screen and a resolution of 1280x960, and a desktop computer with 21 inch monitor and a screen resolution of 1280x960.

In the tests, the participants preferred the KDE "HighContrast Dark Big" theme, but additionally adjusted the font size to fit their needs (22 - 28 pt). In order to get a larger viewport, they set the virtual resolution to 1280x960. That means when moving the mouse pointer to the border of the monitor, the viewport moved till it reached the border of the desktop (see 2.1.5 "Large Font Sizes and Virtual Resolution").

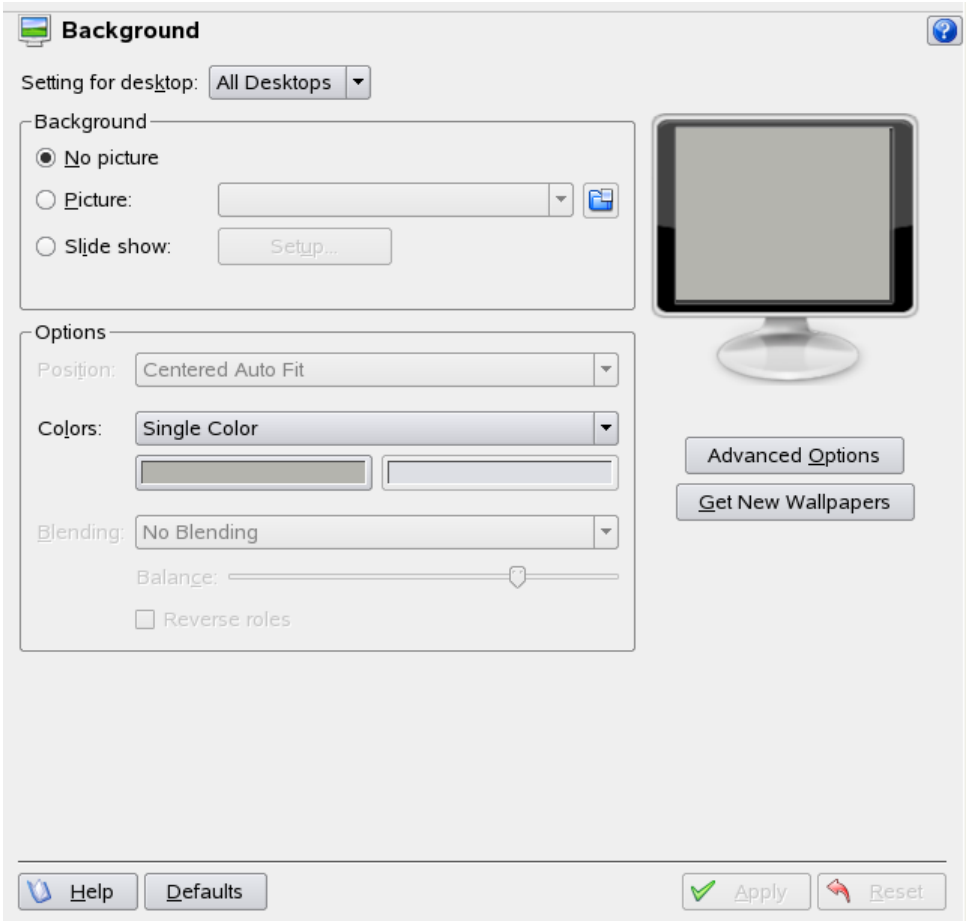
One goal of the tests was to find out if the colour schemes and increased font sizes were useful for partially sighted users as represented by Lars, Christoph, and Mirko (while Mirko usually was not in need of high contrast themes). Another

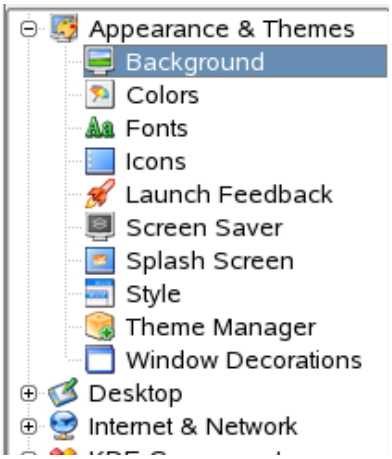
goal was to find out if the schemes were adopted by all relevant parts of the desktop and the important applications.

2.1.1 Basic Characteristics of the High Contrast Colour Scheme

Colour schemes, as part of the KDE high contrast theme, are meant to apply defined colours to all types of interface elements. For partially sighted users, this is of special interest because the colours of adjacent interface elements having a low contrast are replaced by colours having a high contrast. For users who are easily dazzled by bright areas, colour schemes with a black background and white font are optimal ("inverted" schemes).

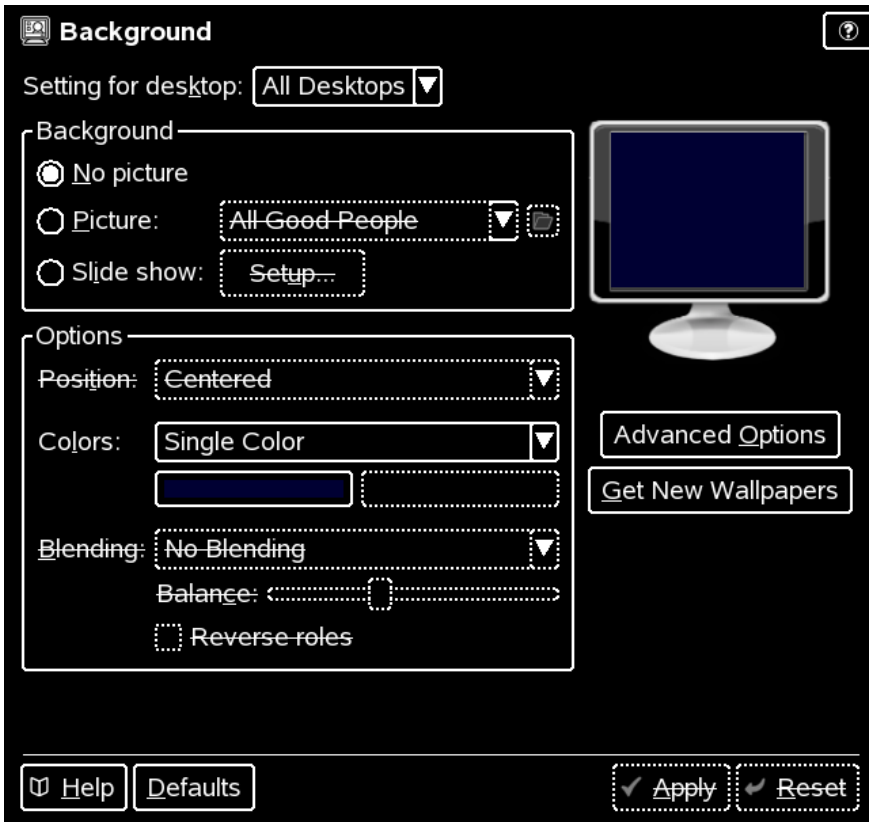
In high contrast schemes, subtle indicators such as grayed out disabled buttons or menu entries cannot be used. Also, normal colour scheme's way to indicate item selection - inverting background and font colour - does not work because the bright background might cause dazzling.

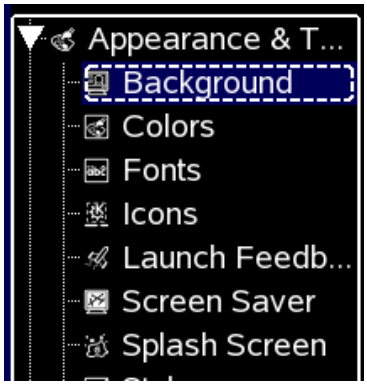




Inactive buttons, drop-downs and checkboxes (above), and a selected item in a tree view (below) in the KDE default colour scheme.

The colour scheme used in the tests handled these problems differently: Regular widgets were black with white text and a solid white outline. Disabled buttons and menu items were surrounded by a dotted line and their text labels were striked out. Selected items in a list, tree or table view were also surrounded by a dotted white line and their background was coloured dark blue.





Inactive buttons, drop-downs and checkboxes (above), and a selected item in a tree view (below) in the high contrast scheme.

Buttons and other widgets in keyboard focus (sliders, menu items, etc.) had an additional dashed border either around the widget or inside it.

Preselected buttons (like "OK" in a dialog) had a doubled solid white outline.

A button in keyboard focus.

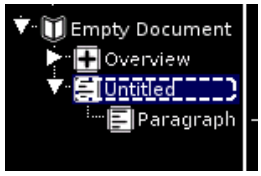
Checkboxes had a square-shaped solid outline and when checked there was a square "dot" inside them. Radio buttons changed their background color to dark blue when the mouse was hovering over them, but otherwise looked as usual, only in black and white.

A checked checkbox in keyboard focus.

Scrollbars had a double vertical line to act as a kind of "guide" for the slider, and a small square to show that the slider was grabbable.

Treeview extenders that expand a tree when being clicked were white equilateral triangles that went downwards when extended and to the right when unextended.

A vertical scrollbar.



Treeview extenders.

Toolbar icons had a white border and a dark blue background when hovered over with the mouse and their text turned light blue. Selected toolbar buttons had a white border and a black background.

Tabs had a white border with a dark blue background when active and a black background when inactive.

The marking of a tabbed layout.

The users got along well with the high contrast scheme. Still, some limitations were observed which will be described in the following sections.

2.1.2 Mouse Pointer

Two of the three partially sighted participants were in need of a strong magnification in order to recognize the elements on the screen. While icons and font could be magnified in an appropriate manner, the size of the mouse pointer itself could not be increased.

The mouse pointer was almost invisible for the participants.

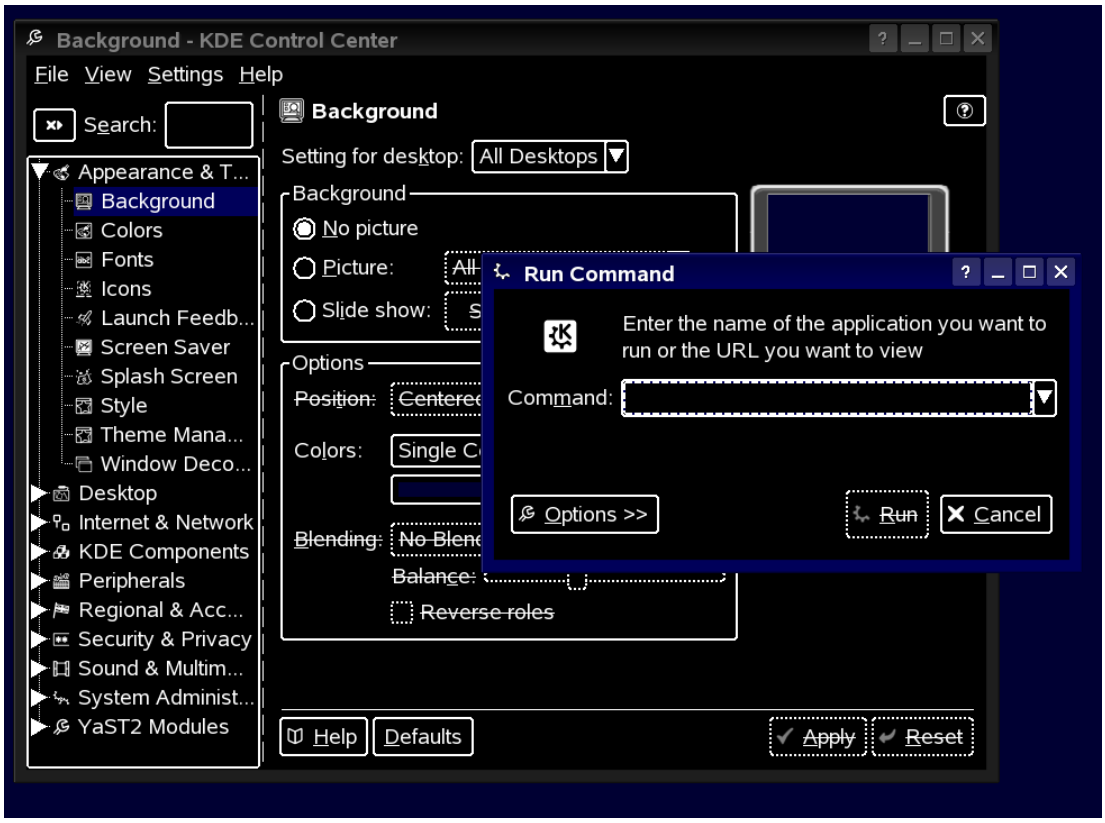
Because of the limited size, the users lost the position of the mouse pointer again and again while working on the tasks. They had to stop their current operation and try to find the mouse pointer. To do so, they moved the mouse in a circle while focusing on a certain position on the screen. When the mouse pointer passed their viewport, they could return to their task.

Returning to their task showed to be another difficulty as the loss of the mouse position also meant losing their position in the application. Due to their limited viewport, each reorientation cost time and caused cognitive load.

All in all, the permanent interruptions caused by the loss of the mouse pointer were evaluated as very disturbing, and significantly decreased the speed of task performance.

2.1.3 Window Decorations

In the colour scheme, active window decorations were marked in dark blue with a relatively thick frame of the same colour around the whole window. Inactive windows had a dark-gray decoration and frame. As both the desktop background and the background of windows were dark, too, the contrast between window decorations and the surrounding areas was extremely low.



Active (blue) and inactive (gray) window decoration and borders.

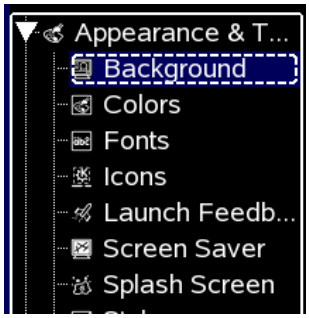
The participants repeatedly had problems to identify the borders of a window which made it hard to find open windows on the screen. This was also a problem when starting an application - as the border was not visible for them, an indicator where the new window had appeared was missing.

2.1.4 Active Interface Elements

Like the decoration of currently active windows, active interaction elements were marked by a dark blue background colour. Without any further indications, this marking was not distinguishable from other elements. This was especially problematic in complex dialogs: In a tabbed configuration dialog, for example, the participants partly did not recognise their current position.

When the currently active element was additionally marked by a stroked line, the participants could distinguish them from inactive elements. Still, they noted that in highly cluttered interfaces, the stroked line would not be eye-catching enough. The complexity of the interface would hide it, therefore additional indicators were be required.

The active tab could not be distinguished from inactive tabs.



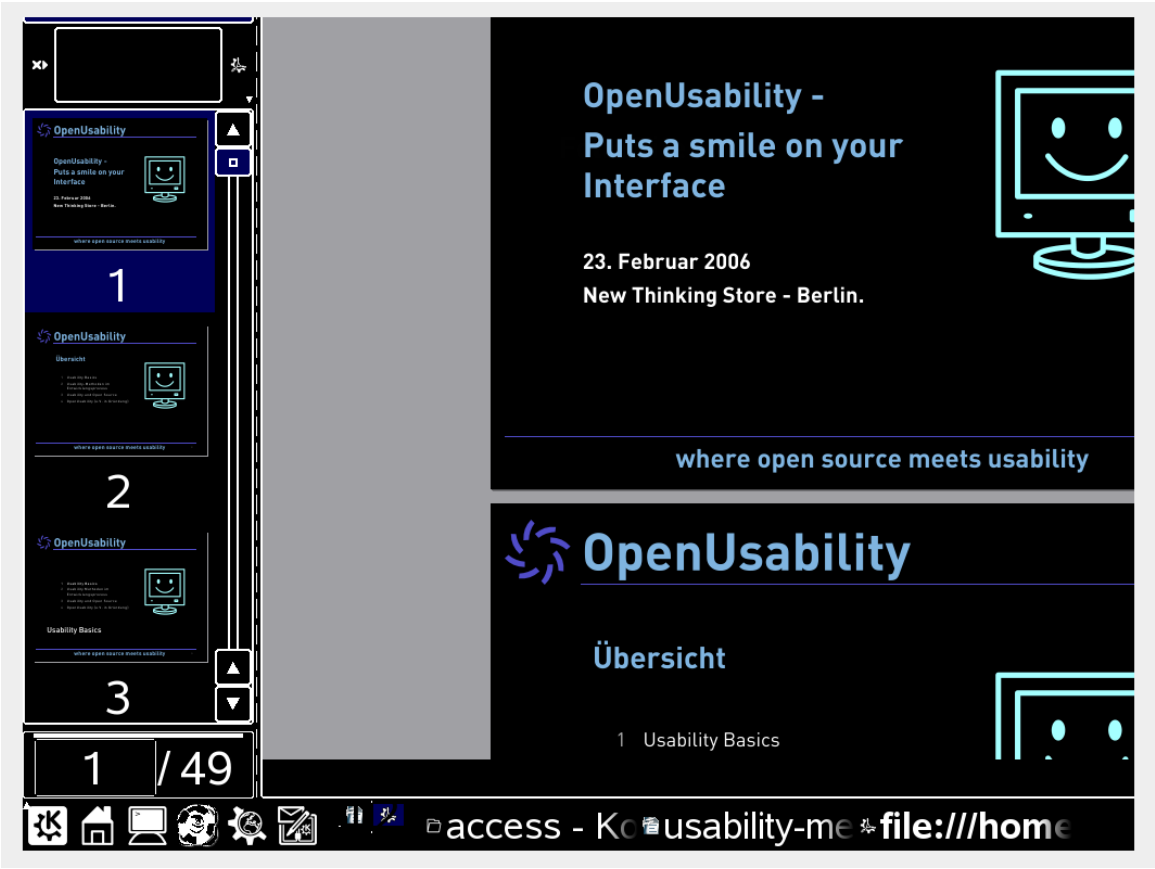
If the active element was additionally marked by a stroked line, the active element could be determined.

2.1.5 Large Font Sizes and Virtual Resolution

Due to the large font size the participants used, the size of windows and dialogs was increased, and all in all less contents could be presented on the screen. With a resolution of 1024x768, the presentation of an email program or other applications that contained many interaction elements and a high amount of text was impossible.

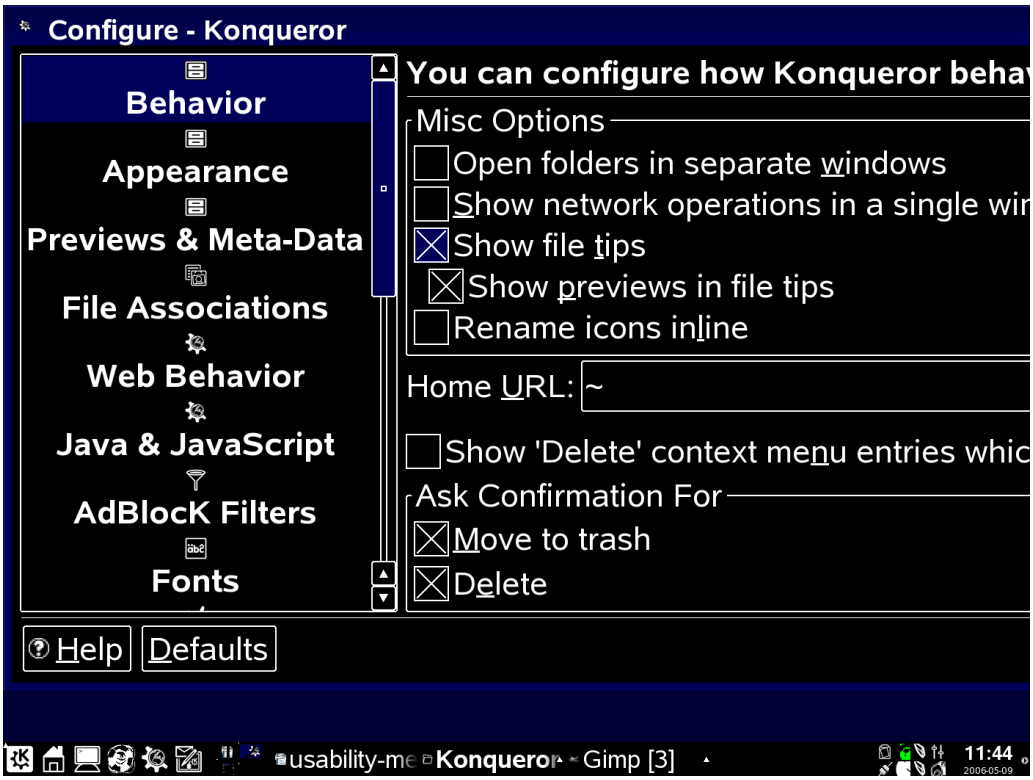
As a workaround, the participants increased the virtual resolution: Instead of the monitor's 1024x768 Pixel, 1400x1050 were assumed. As the monitor could only

show 1024x768 Pixel at a time, the screen always showed just an excerpt of the desktop. When moving the mouse pointer to the right border of the screen, the viewport scrolled to the right. By this, the virtual resolution acted like a screen magnifier – but with some limitations.

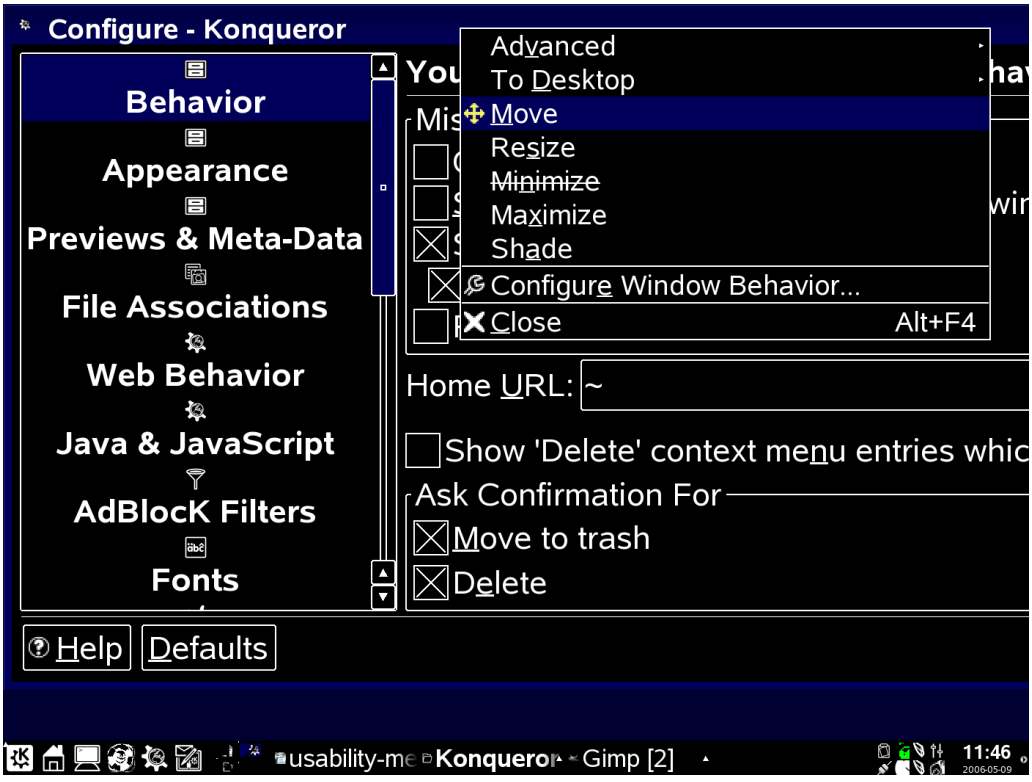


The 1024x768 monitor showed an excerpt of the much higher virtual resolution. When moving the mouse to the right border of the screen, the right part of the desktop was shown. (In the screenshot, the grey border symbolises the monitor border).

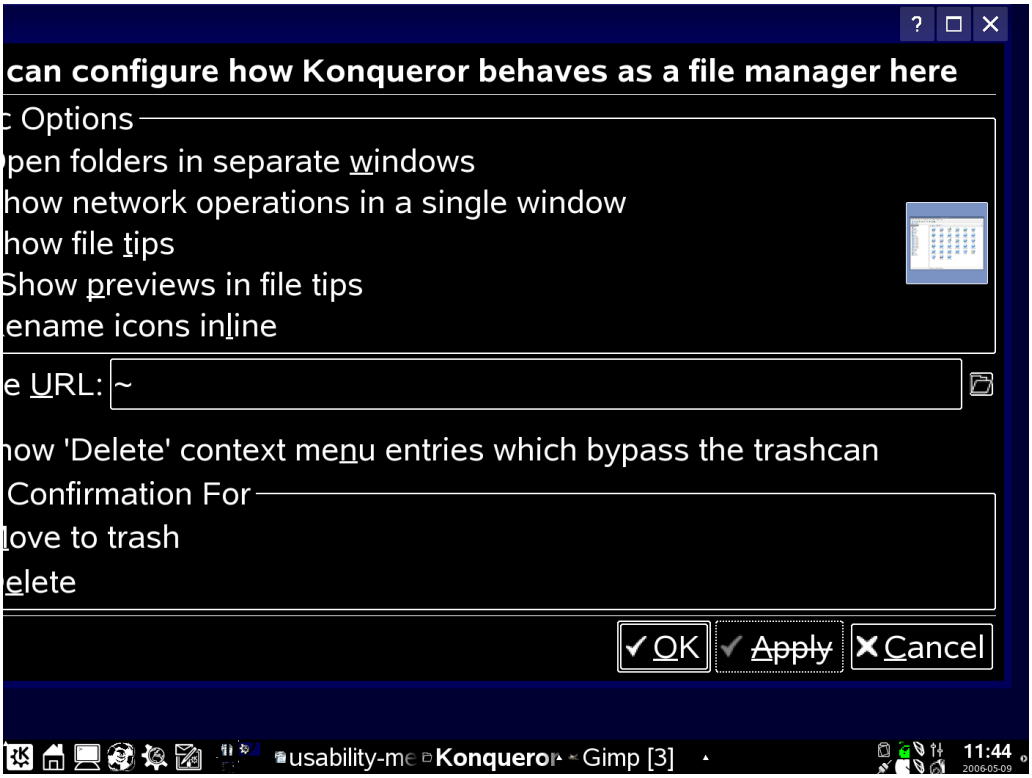
Even if the virtual resolution was increased, it often did not provide enough space to display the whole user interface. This was especially the case for configuration dialogs, where the right border of the dialog often cut off the OK and Cancel buttons. When the users wanted to confirm or cancel a dialog, they had to move the window to the left via the titlebar, then increase the window size on the right until they reached the border of the screen. If the buttons had not yet appeared, they had to repeat those steps till the buttons were visible.



OK and Cancel buttons in this dialog were cut off by the screen border.



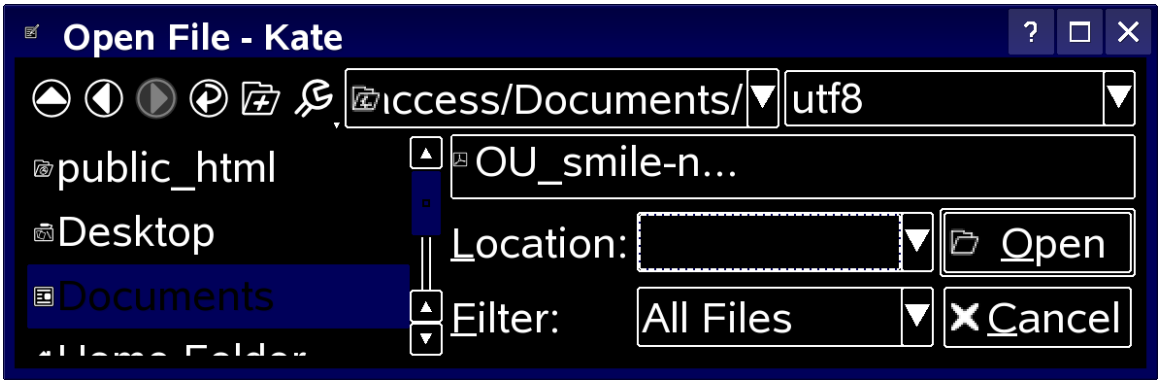
The user then had to move to the window title to move the window.



After two or more iteration steps, the OK and Cancel buttons finally were visible.

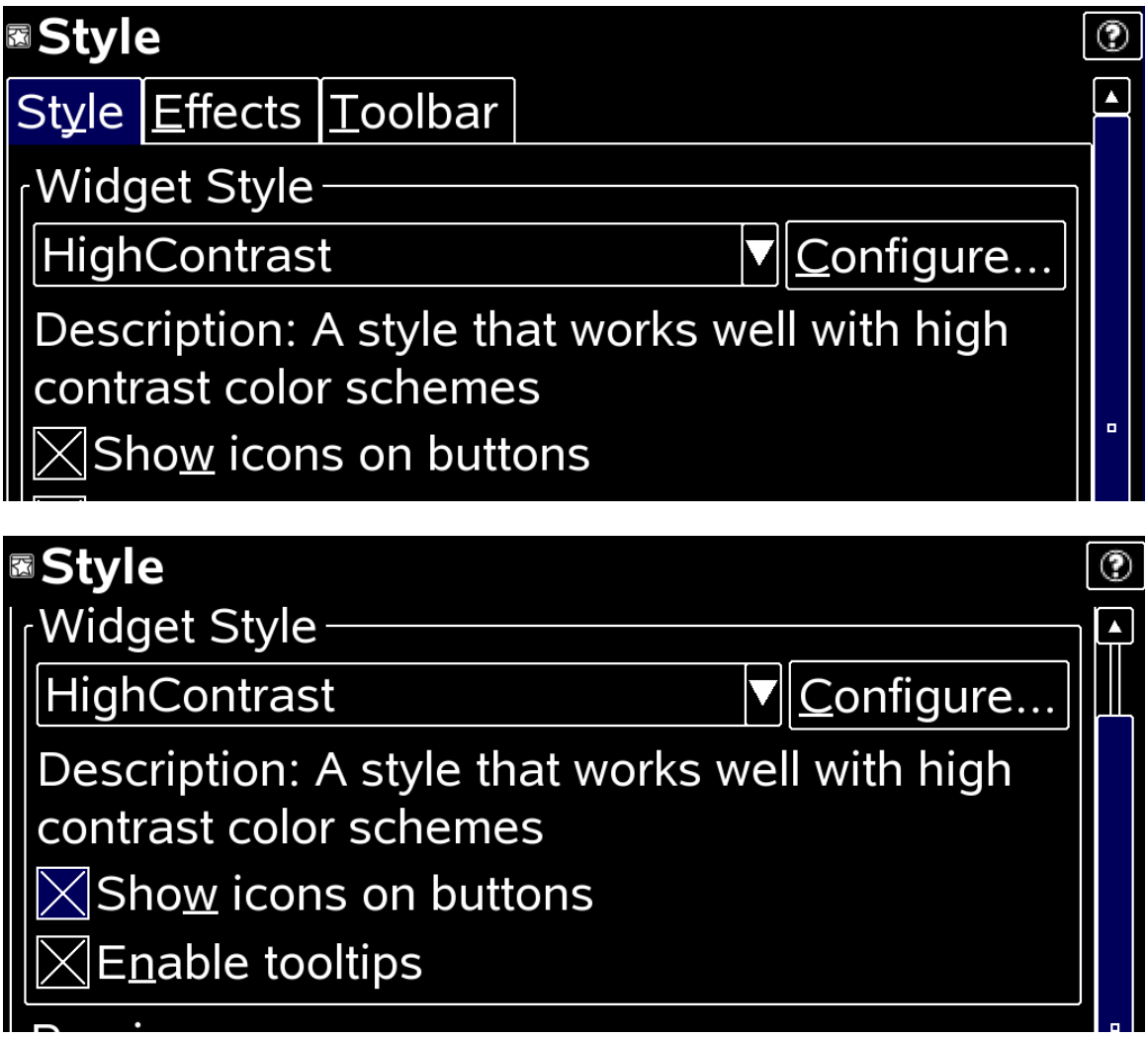
These steps were especially difficult as moving to the title bar and back to the lower right edge of the dialog required a frequent need for reorientation. Given the users' limited viewport, each reorientation cost time and patience.

In complex and text-rich windows such as email applications or file open dialogs, the limitation of total space caused unproportional arrangements of a window's panels: While the left navigation bar in a file open dialog was fully visible, the important file selection panel in the center was squeezed to a size where it could not be used any more.



In some applications, the File Open dialog was distorted at first startup.

Non-conformity with the accessibility guidelines showed to be another problem: In some of the system configuration dialogs in the KDE Control Center, panels did not possess any scrollbars. If the dialog's total length exceeded the screen height, the users had no opportunity to navigate down to the interface elements on the bottom of the panel. When there were scrollbars, it showed to be a disadvantage when the scroll pane embraced a tabbed layout . Being the major means of orientation and navigation, the tabs “disappeared” for the participants when they scrolled to the bottom of a dialog.



Tabs “disappear” for partially sighted users when they are located on a scroll pane as the scrollbar itself is not in the center of the user's attention.

Even worse were dialogs that could not be resized. In two cases, users could not reach the “OK” and “Cancel” Buttons, even after moving and resizing the window. Knowing the common accelerator shortcuts, the two participants pressed “Alt+O” to confirm their changes. In one case, however, this had no effect as there was no “OK” button, but “Close”. The user could only assume this, because he never reached the bottom of the dialog.

On the 21 inch monitor, the users were faced with fewer of the described problems than on the one with a 15 inch size. Still, task completion was hindered by inappropriate presentations and non-conformity with the accessibility guidelines of important dialogs and windows.

Virtual resolution on a 21 inch monitor (KDE running Konsole). The viewport is on the right border of the screen.

2.1.6 Icons

In the test, KDE's monochrome icon set was utilised. Its icons have strong outlines and often resemble very reduced line-drawings. They try to reduce effects like perspective, shading and many different objects in one icon in order to be easily recognizable for people that have problems with low-contrast pictures.

Browser navigation icons from the monochrome icon set.

Document and application icons.

The participants got along well with the monochrome icon set – as one of them stated even better than he had expected beforehand. The simplicity of the icons showed to be helpful.

Still, the monochrome icon set could not cover the whole spread of icons that were used in KDE. The carryover was automatically inverted by the system (todo: how exactly?). While it was evaluated as useful that the icons were integrated with the overall colour scheme, the inversion did not support the perceptibility. Rather, the high number of icons in the interface increased the perceived complexity and distracted the users from accomplishing their task.

2.1.7 Background Images

Some KDE applications make use of fixed background images which hinders the adoption of the overall colour scheme. In the test, this mostly showed to be a problem for startup screens which are shown in many KDE applications, and which are meant to facilitate the orientation: These startup-screens come up at various different occasions, for example in the KDE Control Center, in Konqueror, the file and web browser, and in K3b, the burning tool.

While the font colour was inverted to white in some cases, the use of bright background images made it impossible to read the text. The look of the startup screen could be configured in Konqueror, but in none of the other applications.

Startup-screen in Konqueror. The category heading (next to the items) are not readable as the text is white on a light blue background image.

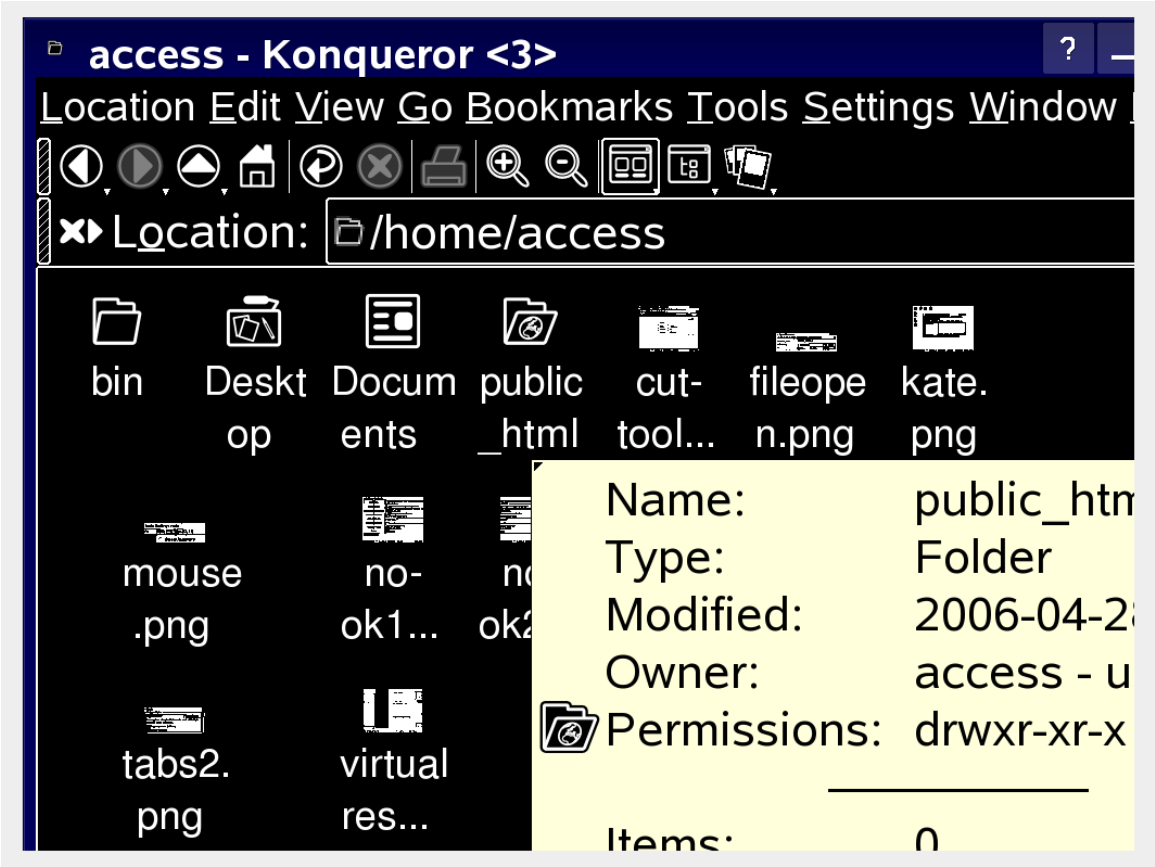
In other applications, fixed background colours ignored the colour schemes: In KSayIt, the second background colour of an alternative row style did not adopt to the colour scheme and resulted in white font on light blue background. Due to the high brightness, the participants were bedazzled and could not identify the contents.

Fixed colours for alternative row styles ignored the colour schemes.

2.1.8 Tooltips

Regarding the overall integration of accessibility features with the desktop, a major shortcoming is that tooltips do not adopt the colour schemes. Especially in combination with icons the participants repeatedly had problems to sound out

the purpose of a certain interface element (e.g. tool in the system tray, button on a toolbar). In addition, tooltips were sometimes cut-off due to their size, and their was no way to move to the center of the screen in order to read the contents.

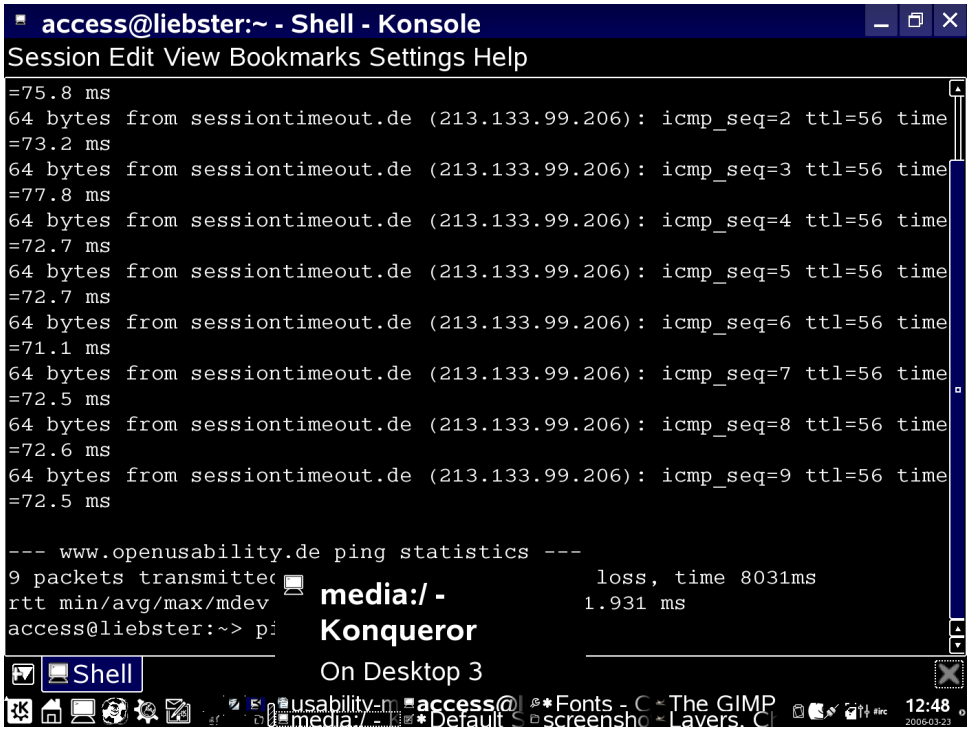


Cut-off tooltip in Konqueror, not adapting to the colour scheme. (In the screenshot, the grey border symbolises the monitor border).

Instead, the lately designed and informative panel tooltips adopted the colour scheme. As the taskbar labels themselves were cut off due to the large font sizes, these tooltips were perceived as a helpful addition to learn about the purpose of a taskbar entry, showing application name, document type, and the virtual desktop it was run on.

Taskbar and tooltip indicating application name, document name (here: title) and virtual desktop.

On the other hand, the tooltips were often accidentally triggered when the users were performing other tasks. For example, when entering commands in Konsole, they accidentally moved the mouse with their arm while typing, and the tooltips showed up. In some cases, they covered commands and output parts of the Konsole. Especially as the tooltips had no visible frame in the dark contrast scheme, understanding why the output had disappeared and how to get rid of the disturbance required another phase of reorientation, and therefore cost time and patience.



Accidentally triggered tooltip covering the current focus of attention in another application.

The nature of tooltips in general – namely to show up when the mouse pointer is

moved over a specific interface element and to disappear as soon as the mouse is moved to another location – has shown to be problematic for high font sizes. In combination with an increased virtual resolution or a screen magnifier which show only a cutout of the actual screen, it happened several times that the tooltip was "cut": Only half of it was displayed in the currently visible viewport..

Due to the nature of tooltips, they disappeared as soon as the users moved their mouse to reposition the viewport. An option to fix tooltips on the screen would be of help here, and would also allow the participants to follow the text with the mouse pointer, their "extended eye", while reading.

2.1.9 Complexity of the User Interface Design

In the tests, it was observed that the more complex a user interface was, the more difficult it became for the participants to identify relevant information on the screen. On the one hand that was due to the limitations of the virtual resolution that arranged dialog panels in an unproportional way and compressed interaction elements inappropriately. On the other hand, an increased number of interface elements created visual clutter.

Due to their handicap, the participants' viewport was limited in size. They could not gain an overall view and spot the interface element in question, but had to go through the interface sequentially. An increased number of elements or a high complexity of the layout (nested, tabbed, etc.) affected the time and resources that were required in order to accomplish a task.

2.1.10 Adoption by the Applications

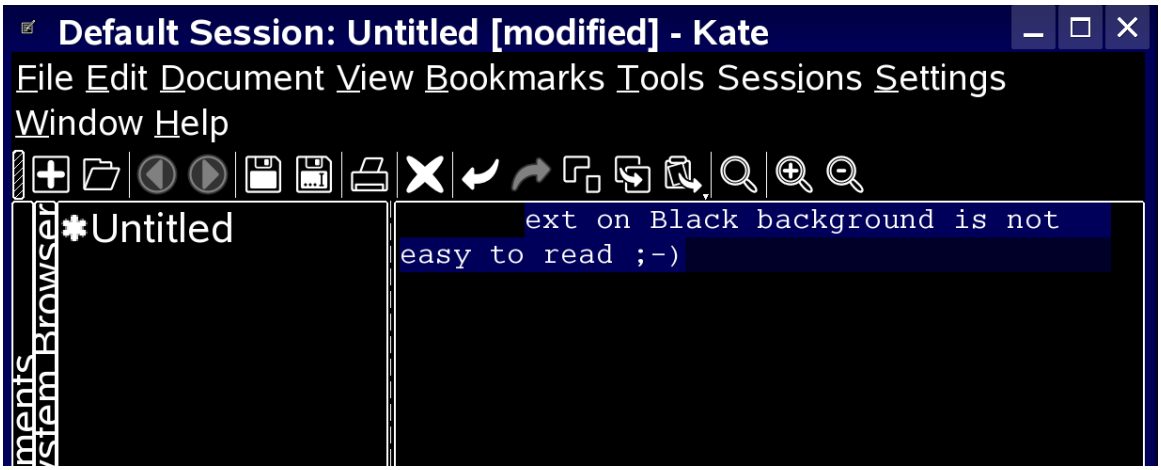
An important factor regarding the usability of a specialised colour scheme is its adoption by the desktop and the applications, both regarding user interface elements and the contents, for example in text editors or displayed documents. After switching to the high contrast theme, the users had to log out and in again to gain a complete adoption of the colour scheme.

While the missing adoption of colour schemes by tooltips and the problem of background images was described above, this section gives special attention to specific applications.

With regard to the interface elements, most KDE applications had adopted the colour scheme after the re-login. The look of non-KDE applications, for example OpenOffice.org or Firefox, could not be manipulated by that means - even if OpenOffice.org is supposed to automatically adjust to the colour settings of the operating system. However, there is an option to create customised colour schemes in OpenOffice.org.

With regard to the contents displayed by an application, the majority of KDE applications adopted the high contrast colour scheme automatically. Still, in the

test there were key applications that required a manual intervention: Konqueror and KPDF, for example, did not automatically adjust their contents to the colour scheme. The text editor Kate adjusted the background colour to black, but kept the default font colour - which was also black.



Per default, Kate shows black text on black background when converting to the high contrast dark theme. Only when selecting text, it becomes visible.

This customisation was easy in some applications, and rather difficult in others. All in all, a consistent way to adjust contents of an application to the system colour scheme was missing: In KPDF, the colour scheme could be set in the tab “Accessibility” in the settings dialog, in Konqueror, he had to choose the tab “CSS”, and in Kate, there was a tab “Fonts and Color Schemas”.

While it is reasonable to ask the user for a manual configuration according to his preferences (should an image viewer show images in high contrast or in real colour?), the inconsistent location of these settings hinders partially sighted users significantly to customise their applications.

2.1.11 Conclusions Regarding the KDE High Contrast Theme

The users much valued the high contrast colour themes in KDE. Especially the consistent application of dark backgrounds in combination with light font was appreciated. In many other colour schemes and when using inversion mechanisms of a screen magnifier (see 2.2.3 “Invert styles”), selected text often is also inverted, resulting in a bright background and black font colour which was not readable for two of the test participants. Furthermore the high contrast icons were mostly perceived as helpful.

Lars, who made use of it in his daily work, claimed that the KDE theme provided him with the highest possible flexibility to adjust the desktop to his personal needs. Being able to set each colour of the interface elements individually, he

could overcome problems with inconspicuous window decorations or selected text. He also customised the applications he usually made use of to apply to the overall colour scheme, and added a larger mouse pointer to the system settings. Still, one should keep in mind that Lars is a student of computer science, that means he is highly skilled regarding computers and possible settings. With the given defaults, it is likely that less skilled or less explorative users would have a hard time to fully adjust the desktop to their needs.

Another problem that needs to be addressed is the consistent implementation of accessible interface elements - configuration dialogs whose OK and Cancel buttons can not be reached in huge font sizes because of a fixed dialog size or missing scrollbars are one example, tooltips that do not adopt to colour schemes and are not completely displayed another one. Keeping dialogs simple is a requirement that does not only account for creating accessible applications.

Weaknesses of enlarged high contrast themes in general come up when using extremely large font sizes: It is unavoidable to be faced an unproportional enlargement of dialogs, resulting in cut-off labels or hidden interaction elements.

A problem for users with a lower visual impairment, the definition of a fixed high contrast theme including font sizes may be inappropriate as they may need varying magnification levels and wish to use different colour settings depending on the context of use: Images, for example, might best be seen in real colour, while text documents should be inverted. Changing colour schemes and font sizes “on the fly” is not possible in a X-based desktop system.

2.2 Screen Magnifiers

Screen magnifiers address the problems of a fixed colour scheme, fixed font sizes and unproportional enlargement of dialogs by zooming into the desktop. Instead of changing the system settings, parts of the screen are picked and magnified.

In the session, the development version of a new screen magnifier for KDE was tested on a notebook with a resolution of 1024x768. The screen magnifier had a number of view options: Firstly, the user could set the magnification level. Secondly, the user could select between a fisheye and a fullscreen mode. In fisheye mode, the whole screen was visible at a time, but only the center was zoomed in. To the borders, the view was compressed. In fullscreen mode, the whole screen was equally zoomed in, but the user could only see the section displayed on the screen.

Thirdly, the inversion style could be selected. There were three different types: One inverted the brightness, one had a dark outline, and one was inverted and had a dark outline. Alternatively, the normal colour style could be maintained.

Fourthly, there were two different magnification algorithms, a pixel repeat algorithm and a normal one.

2.2.1 Zoom Factor and View Modes

Before starting with the actual tasks, the properties of the screen magnifier were adjusted to the users' preferences. One partially sighted user, Mirko, preferred a magnification level of 1 which corresponded to a zoom factor of 1.25. The two others, Christoph and Lars, preferred level 5 which corresponded to a three fold magnification.

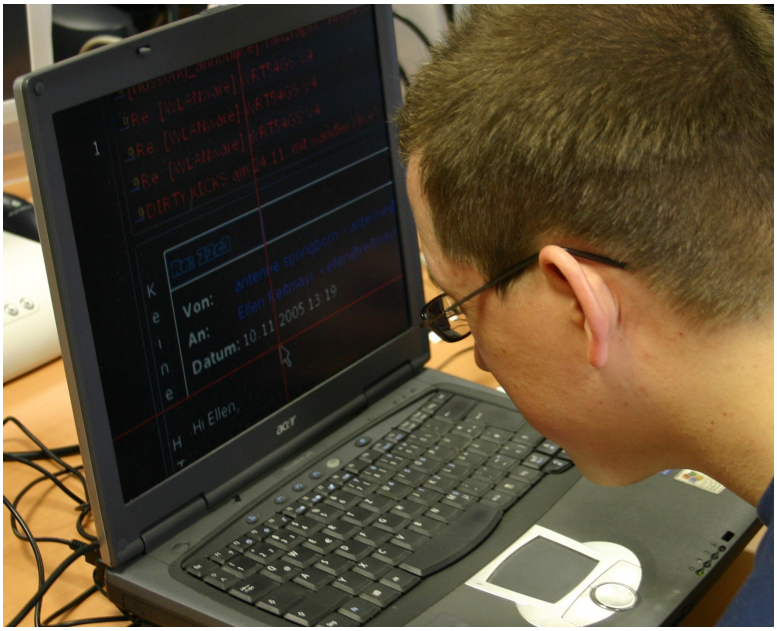
Then, the two view modes were probed. In high zoom grades, the borders of the desktop were strongly compressed in fisheye mode on a 14 zoll monitor. Lars and Christoph therefore preferred the fullscreen magnification.

With Mirko's comparably low magnification, fisheye mode provided the benefit of gaining an overview over the whole desktop while interacting with a certain application. In the first run, he therefore selected fisheye mode.

Regarding screen magnifiers, Christoph and Mirko were experienced users, while Lars normally made use of an increased virtual resolution. At home, Christoph usually switched quickly between different magnification sizes, depending if he needed an overview over the screen or worked with a certain application. To switch the zoom factor, he had a special button on his mouse.

2.2.1.1 Fullscreen View Mode

Fullscreen magnification is an option to surpass the limitations of increased font sizes in combination with a virtual resolution: Instead of making the system increase its appearance, the magnifier cares for it. The system settings stay unaffected, and problems as described in 2.1.5 “Large Font Sizes and Virtual Resolution” are bypassed.



Christoph using the screen magnifier in fullscreen mode.

As an early development version of the new KDE screen magnifier was used, the objective of the test was to learn about the target user's preferences regarding magnification types and their behaviour. In a feedback process, the user wishes and requirements should be integrated with the tool.

A major problem the participants experienced with regard to the screen magnifier was that when moving the mouse over the whole desktop, the viewport did not follow fluently but it “jumped”. For example, when moving to the right border of the monitor, the viewport was abruptely moved to the right, adjacent position of the screen. The mouse pointer was repositioned into the center of the monitor.

This behaviour was not appropriate for the participants by two reasons: Partially sighted users require a strong guidance leading them along the applications and their contents. As one of the participants stated, the mouse pointer functions as their “eye”, moving over the screen. While it is important to keep track of this extended eye, it is equally important that the eye keeps track of its current context, that it smoothly follows the contents.

In the given screen magnifier, the users lost both: The current context as the focussed content “jumped” somewhere to the left when reaching the monitor's right border, and at the same time they lost the mouse pointer - their extended eye - when it was abruptely relocated to the center of the monitor.

Reorienting and gaining focus of the new pointer position was the first required step to continue reading. Finding the proper vertical position in the left part of the new viewport and remembering the contents they had last seen was the second, even more distracting step.

When reading an Email, for example, it regularly happened that they lost their

vertical position and read the same line twice or three times because the mouse pointer happened to be a bit higher or lower than the actual row they read. While that problem also exists in more fluent magnifiers, the abrupt changes especially hindered continuous reading. As a workaround, one user remembered the line number of the row he was just reading in a given paragraph. To proceed with the next line, he first counted down to the previous line, then started to read the next one.

The participants evaluated this kind of behaviour as very disturbing. They asked for a smooth motion, allowing them to follow along the applications and contents by his own speed, is a requirement that needs to be met for full screen magnifiers.

2.2.1.2 Fisheye View Mode

When asking the participants for the expected benefits of a fisheye magnification, they stated it might be helpful to gain an overview over the whole desktop. A typical problem in high zoom levels was, for example, that the viewport was focused onto an application while suddenly it stopped to react on mouse or keyboard input. Usually, that was because a modal dialog had popped up at another part of the screen. With a fisheye magnification, it might be easier to realise that something has changed outside the focused viewport.

[foto/screenshot: close-up of fisheye]

All in all, they expected an improved overview over events that happen outside the current viewport, for example notifications from an instant messenger, new mail notifications, or windows that open at locations they did not expect.

Mirko using the screen magnifier in fisheye mode. The borders of the screen (right and bottom) are compressed.

With the given monitor, fisheye could not be applied usefully for high zoom factors as the compression on the borders was too extreme. Therefore, extensive testing was performed with the user who required less magnification only.

In the test, some situations approved those expectations: When interacting with two applications, fisheye facilitated the overview over the adjacent windows, and a directed navigation towards relevant interface elements outside the zoomed viewport was possible. For example, having an email read aloud (see 2.3.3 “Reading via the Clipboard”) required the operations in two applications and in the system tray. First, text had to be selected in a mail. Then, the user had to navigate to the system tray and activate Ktts (KDE Text to speech application) which was docked there. Finally, he had to make settings in the Ktts operator window, that popped up in the middle of the screen. Without the overview over the whole desktop as provided by fisheye, this task would have been much more difficult.

Other situations were less clear of an advantage. When reading an email, part of the text in a row was compressed. There was no clear judgement if this behaviour was preferred to a fullscreen magnifier – in both cases, the viewport needed to be moved over the screen. Still, Mirko stated that fisheye might be an advantage when reading long text documents.

Even if most of his expectations regarding fisheye were fulfilled, some other,

non-expected problems came up: As in fullscreen mode, the viewport jumped to the adjacent position as soon as the user touched the sensitive part around the zoomed area with the mouse. But as the zoomed area was significantly smaller than in fullscreen mode – only one third of the screen was fully zoomed, the rest showed the compressed borders of the desktop – the magnifier was perceived to be very sensitive. Minimal movements with the mouse caused the viewport to jump to another position, which created an uneasy use experience. On the one hand, this was due to the small monitor used in the test, on the other hand an indicator where the sensitive area started was missing. In fullscreen mode, the border of the screen was the obvious indication of the sensitive area - here, there was none.

As in fullscreen mode, the magnifier's behaviour when switching the viewport made it difficult to keep the current position on the screen. Directed navigation, for example in nested menus where parts of the submenu were outside the fully zoomed area, became extremely difficult by that means: When reaching the item in need, the viewport jumped to a new position, the mouse pointer was positioned in the center, and the menu was closed.

Several solutions were proposed regarding these problems: First, a clear indication of the sensitive area was asked for, for example a red border. Second, a more fluent motion of the magnifier was wished. Third, the viewport should also follow keyboard strokes. Fourth, an additional option to slow down the mouse motion when navigating in difficult areas, for example nested menus, would be of help. All in all, the navigation would have been easier on a larger screen, where the fully zoomed area would have been bigger.

2.2.2 Mouse Position and Movement

In the test it has turned out that the positioning and repositioning of the mouse is a factor that is crucial for the usability of a screen magnifier.

As mentioned above, the mouse pointer functioned as an extension of their eyes for the participants - losing sight of it required reorientation, an act that unnecessarily consummated cognitive resources and distracted them from their actual task.

[two fotos/screenshots: mouse at right border, then mouse in center and new viewport]

In the development version of the screen magnifier, the mouse pointer repositioned itself automatically in the center of the screen as soon as the user reached the border of the monitor. While it was suspected that a centered mouse pointer might facilitate orientation for the users - they would not have to search for it but simply needed to move their eyes to the center of the screen - the participants did not appreciate the given behaviour: The abrupt movement of the mouse pointer made them lose sight of it and of the current position on the screen they were just about to look at. The participants were faced double efforts

when returning back to their task: First, they had to get track of the mouse pointer, then locate their previous position on the screen, for example a sentence in an email they were just about to read.

While both Mirko and Lars expected the mouse pointer to remain at the position on the right, the displayed screen smoothly moving in, Christoph later explained that he actually appreciated a centered mouse position. He would often use that mode in the Windows screen magnifier TextZoom. There, the mouse position could be kept in the center of the screen permanently. Instead of moving the mouse to the border, the desktop was moved below the mouse until it reached a border. This mouse behaviour allowed him to fully concentrate on the displayed screen below the mouse, and he never ran in danger to lose track of the mouse or the currently focused content as both were located in the center of the monitor.

Concluding, it is crucial that the mouse pointer can easily be spotted and supports a fluent view onto the displayed screen. Abrupt interruptions, both regarding the mouse position and the repositioning of the view port, hinder the user's work flow.

2.2.3 Invert styles

The screen magnifier offered three different invert styles: The “invert” style simply inverted the brightness. “Inverted dark outline” additionally painted a dark outline around the light areas. The “dark outline” painted a dark outline around bright areas but did not invert the screen.

[todo: Gunnar, please check description and add fotos/screenshots]

Mirko did not need to set an invert style. The two other partially-sighted participants both preferred the “inverted dark outline style”.

But when reading mail in the email application KMail, both experienced problems: They were faced blue and red font on black background as the style inverted brightness but did not consider colour saturation. The blue mail sender in an email header could therefore not be identified by the two users. Similar problems occurred during spell-checking in a word processor. Providing an invert style that inverts all colour channels instead of only brightness might be of help here. An additional increasing of the contrast was a wish one of the users named repeatedly.

Another problem came up when the original background was dark and the font was white, as for example in the case of a selected email in the inbox. Inverting resulted in dark text on a white background – a situation that was handled properly in the colour scheme, but can not be considered by a screen magnifier. In this situation, the possibility to quickly change between different invert styles via shortcuts is essential, so users can change to a view mode that better supports their needs.

2.2.4 Mouse Recognition

A typical problem for partially sighted users is that they lose the current position of their mouse. In order to facilitate the reorientation, the screen magnifier marked the mouse pointer with a red cross that reached the borders of the screen. The red cross was helpful for two of the participants.

Still, they missed an opportunity to change the colour of the mouse pointer itself to increase its contrast. Especially when the mouse pointer changed its shape over links, the participants almost did not recognise it due to the lower contrast.

2.2.5 Focussing Windows

Another point related to repositioning and finding relevant information on the screen was the handling of windows requiring the focus. In the tests, it happened several times that users committed an action that caused a new window to open: When writing a mail, for example, the users clicked the toolbar icon "New Mail". While the mail editor window actually appeared, the users did not realise that as it was opened outside their current viewport. They clicked the toolbar button two or three times, till they finally started to search the screen for the new window by moving the viewport along the desktop.

[screenshot: viewport when clicking new mail icon (composer window not visible)]

A similar problem came up when one of the users wanted to have text spoken by the text to speach application Ktts. It was docked to the system tray, and a left-click onto the icon opened the manager window. The user navigated down to the system tray and clicked the icon, but nothing noticeable happened. By right-clicking he opened the context menu, but there was no indicator how to restore the manager window, because the corresponding menu item changed to "Minimize" when the window was displayed on the screen. In the task bar, there was no entry for Ktts, and when searching for the window it could not be found as it was hidden by another window. The participant double-clicked the item, left-clicked it, and finally stumbled over the entry "Restore" in the context menu when the Ktts window was currently minimised due to the prior double click. After restoring the window by that means, he searched the screen in a directed way until he found the manager window.

An item to fetch the Ktts manager window to the front ("Restore") is only available when it is closed.

The problems described in the last paragraph are due to a combination of several factors: First and importantly, the screen magnifier should inform the user about windows requesting the keyboard focus, and should smoothly move there.

Second, it clearly shows some shortcomings of the KDE desktop, mostly due to inconsistencies: When left-clicking an item in the system tray, a window may be shown, it may be hidden, or no window may be shown at all (e.g. changing keyboard input versus a menu is shown as for KPowersave). The icon itself does not provide an indicator of the upcoming behaviour.

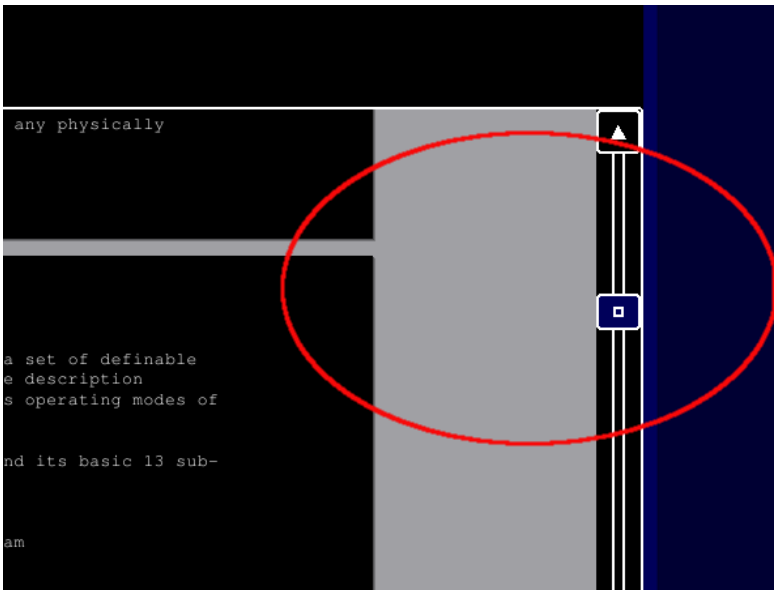
For partially sighted users, knowing that specific interface elements behave in a defined way is especially important as every reorientation requires efforts to adjust to the new conditions. This act of reorientation distracts from completing a task and hinders information assimilation.

2.2.6 Focus in Applications

A similar issue was following the focus within an application. In the development version of the screen magnifier, the viewport followed the location of the mouse pointer only. When the user typed text, for example in an email or in a text editor, they had to relocate the viewport manually with the mouse when the typed text exceeded the limits of the current viewport. The reorientation again cost cognitive efforts and hindered the information assimilation. However, this behaviour is due to the early development stage of the screen magnifier and can easily be overcome by listening to the type of active input and making the viewport follow it.

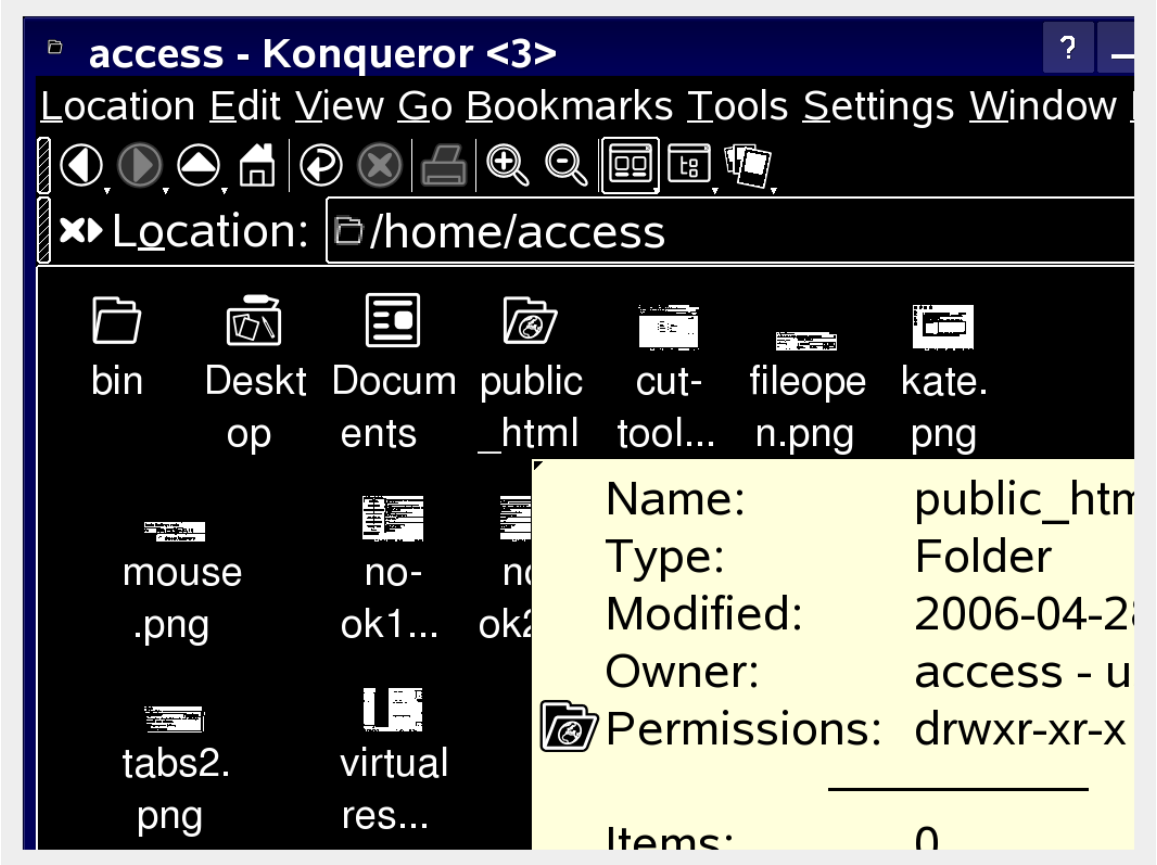
But even here an important aspect needs to be considered: In the past, it had frequently happened to one of the users that unintended mouse movements, for example when the table wagged for unknown reasons, "stole" the input focus from the keyboard and allocated it to the mouse in another screen magnifier. To keep on typing, he then needed to manually set the mouse cursor to the previous text position. A mechanism that differentiates unintended mouse motions from directed ones would be helpful.

In the development version of the screen magnifier, using the scroll wheel also was not yet considered. It has shown that the mouse's scroll wheel is a crucial tool to navigate in applications: If the user needs to utilise the scroll bar to go down to a certain position of the page, the viewport is focused onto the scrollbar. For large zoom factors, the document contents are outside the visible area, so the user has no indicator how far he has scrolled so far. Apart from supporting mouse scrolling, a more sophisticated scrollbar design might be of use.



In high magnification levels, the viewport did not reach the headings of a document. The user's point of attention may be even smaller (red circle).

An unexpected problems was caused by tooltips: When a tooltip appeared close to the border of the viewport, it sometimes happened that its ends were cut off. This caused a paradoxon: When moving the mouse to change the current viewport to read the hidden part of the tooltip, it disappeared. When not moving the mouse, the tooltip was incomplete. In applications where tooltips provide detailed context information, it is almost impossible to read them completely as their size exceeds the viewport.



A cut-off tooltip in Konqueror (screenshot taken with colour schemes, not screen magnifier).

2.2.7 Magnification Algorithms

To magnify the size of interface elements, the users had the choice between two different algorithms. The normal algorithm functioned as standard screen magnifiers, the pixel repeat algorithm had a different smoothing algorithm [todo Gunnar: please check and provide a better description].

In the first run, neither of the three partially-sighted participants recognised a difference between the two magnification algorithm. Only when it came to the German special characters “ä”, “ö” and “ü” the participants experienced a superiority of the pixel repeat algorithm. Here, the dots on top of the letters could be identified in a better way which was important to quickly understand the meaning of certain words.

2.2.8 Conclusion Regarding Screen Magnifiers

Even if a development version of the screen magnifier was tested, the advantages compared to fixed font sizes and an increased virtual resolution became obvious: While interface elements and dialogs were cut off in the latter case, the screen magnifier provided a clear view of the whole screen without a need to change window sizes in order to get the "whole picture".

For smaller magnification levels, the fisheye mode showed to be valuable to gain an overview of the desktop. In combination with a large computer screen and a high resolution the benefits should become even more obvious.

An idea to gain an overview of the desktop and came up during the tests was to split the screen and fix important parts of the desktop. One might, for example, keep taskbar and system tray on a fixed position on the screen for a faster navigation, and use the rest of the screen as working area. For users that require high magnification levels, this might be an alternative to fisheye mode.

To increase the usability of any view mode, it is crucial to support a smoother movement of the viewport when moving with the mouse over the screen. Optionally, one might provide a mode where the mouse pointer is fixed in the middle of the screen while the screen is moved under the mouse.

Also, the finished version of the screen magnifier should determine if the currently active input device is the mouse or the keyboard. The viewport should always follow the active one. Importantly, accidential movements of the mouse should not distract the viewport from following the keyboard input.

When windows gain the focus, the viewport should move smoothly to the position of the window and center it on the screen.

Offering invert styles by the screen magnifier instead of – or in addition to – colour schemes has the advantage that all objects displayed on the screen can be inverted, including PDF or images. Still, the participants missed an opportunity to increase the contrast, and an option to consider colour channels in the inversion.

The appearance of the mouse pointer itself should be configurable to a more eye-catching colour and a higher contrast to better visually guide the users.

In the tests, the necessity of shortcuts for the different magnifier functions became obvious: To optimise the flexibility of the screen magnifier there should be (configurable) shortcuts to switch between different levels of magnification and invert styles. Such switches are useful, for example, when the user changes between watching images (real colour) and reading text (high contrast, large zoom factor).

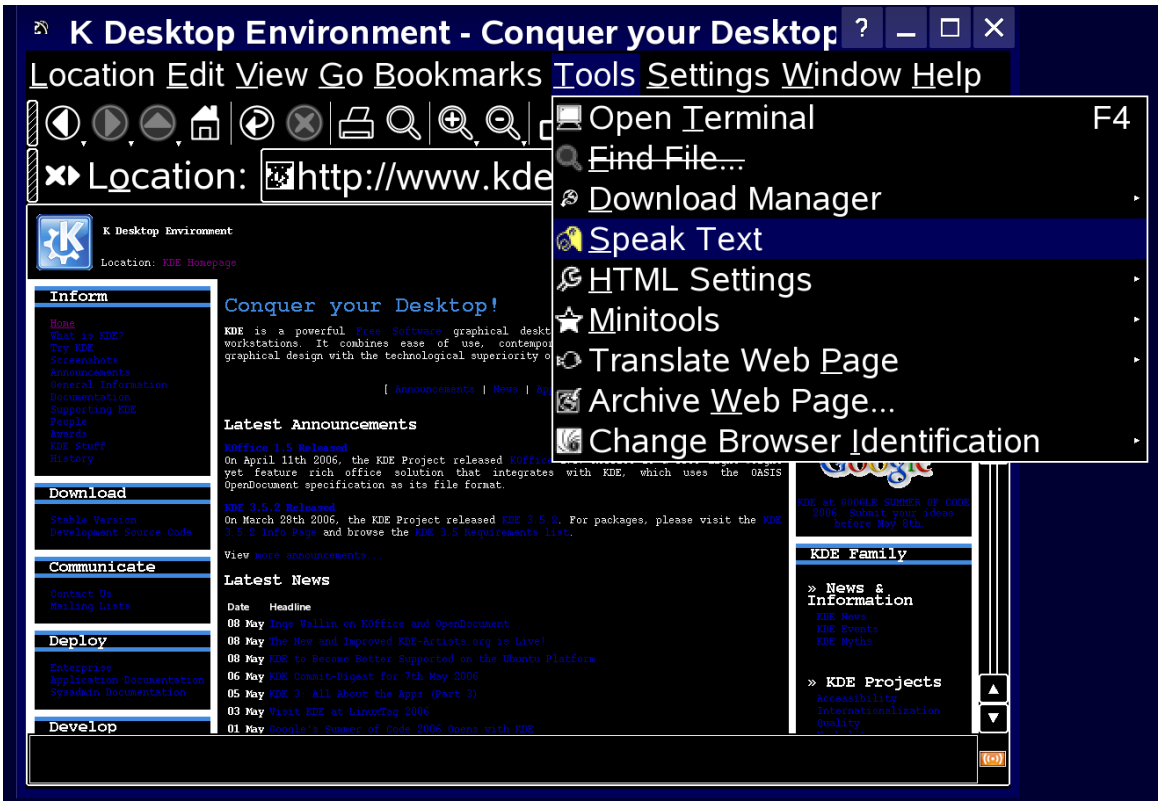
2.3 Document Readers

In KDE, the text to speech system Ktts to read a document's text is integrated with different applications in different ways. In the test, four types of integration were evaluated in order to learn about advantages and disadvantages of the different approaches:

Firstly, text was selected and the text to speech (tts) system was initiated via the menu (Konqueror). Secondly, text was selected and the tts system was initiated via the context menu (KPDF). Thirdly, text was selected, copied to the clipboard, and read by Ktts when selecting the proper menu entry of Ktts' system tray representation. Fourthly, a file was loaded into KSayIt, an external application loading complete files and reading them.

2.3.1 Reading via the Menu

To have a web page read aloud in Konqueror, the user can go to the “Tools” menu and select the item “Speak Text”, which was evaluated as intuitive. If the user selected text before, only that text was read, otherwise the whole web page was read aloud.



In Koqueror, Ktts integration was located in the Tools menu.

In the test, the users were instructed to read the web page of KDE (www.kde.org). Two of them did not select any specific text, so the whole web page was spoken. As Ktts goes through all interface elements, it also reads the contents of drop-down menus or other hidden information. In the test, one of the first elements on the KDE web site was a drop-down menu to switch to another language, holding approximately 30 items. Before it started with the actual text of the web page, the participants therefore had to listen to a row of unrelated words. This was more difficult than expected as there was no item in the "Tools" menu to stop the speaking. Closing the Konqueror window was not of help either – the voice kept enumerating different country names.

Contents of the language and location drop downs at the beginning of the KDE Homepage (excerpt from the "Current sentence" display in the Ktts manager window).

From preceding tasks, the participants knew that Ktts was docked into the system tray, and that a manager window could be opened there. They went to the system tray, opened the manager window – each of which cost time and efforts because it required reorientation – then started to search the manager window for speech controls. As the window was quite complex (a tabbed layout with seven tabs, each of them holding several input widgets), it took another three to five minutes to find the appropriate button to stop the "job".

The complex dialog layout made controlling the text to speech jobs difficult – especially in a three-fold magnification.

Another user had selected text before he started Ktts and was faced a different problem: The marking for selected was blue and gave little contrast to the black background. He could not be completely sure if he had selected the whole paragraph or just a part of it.

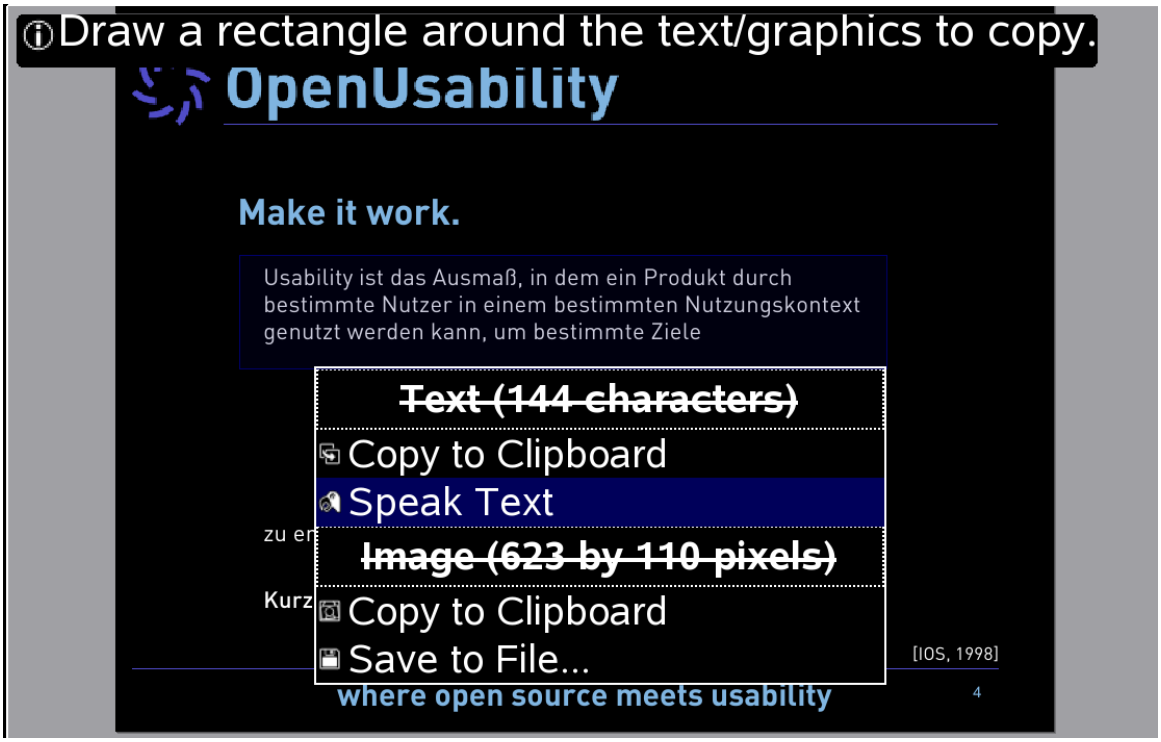
2.3.2 Reading via the Context Menu

In KPDF, the KDE PDF reader, the participants were asked to read the contents from a slide in a PDF presentation. In KPDF, it was not possible to have the complete document read, but the user had to select text, then call the context menu and select the item “Speak text”. As in Konqueror, this was accomplished while using colour schemes, large fonts and an increased virtual resolution.

Here, all three participants had severe problems to accomplish the task. First, one participant stumbled over the location of the function. As in Konqueror, he expected a menu item that would read either the whole document or a user-defined selection of pages. After having searched for approximately ten minutes, he decided to select text first.

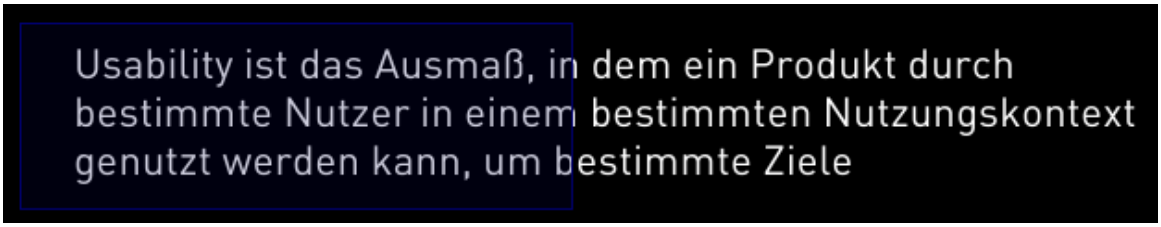
Text selection was the second major problem: To select text, the users first had to switch into text selection mode. This could be done via a button on the toolbar or the menu (“View” → “Select Mode”). Neither icon nor label were enough self-

explaining. The shortcut “Ctrl-A” did not work to select the contents of the whole document.



In the KDE PDF reader, marked text could be spoken via the context menu.

In text selection mode, the users had to draw a rectangle around the text in question. Other than in Konqueror, there was no row-oriented selection, but a graphical one. The tool selected exactly the words within the rectangle, that means if a user did not draw the rectangle to the right end of a paragraph, the words outside the marking were not read. Due to the high magnification level, the users had to scroll to the right in order to mark the whole width of a text paragraph – as apparently needless efforts, they mostly skipped the scrolling. As a result, incomplete sentences were read aloud, and the users had no chance to understand the meaning of the text.



The above selection was spoken as follows: “Usability ist das Ausmass, in bestimmte Nutzer ine einem genutzt werden kann, um b”.

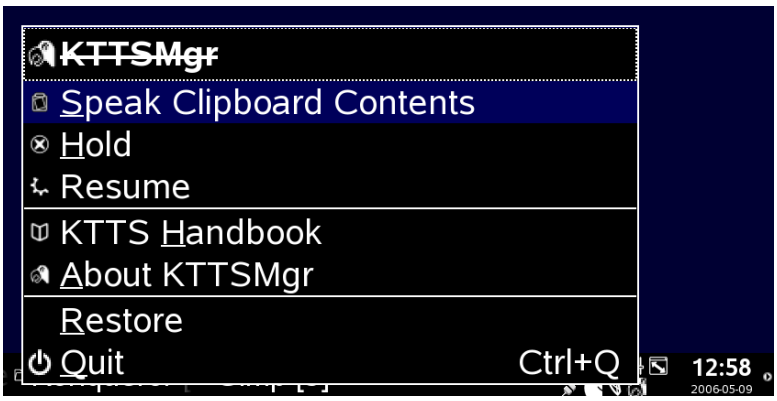
Additionally, the visual marking was even less clear than in Konqueror: Only a very thin line was drawn to indicate the selection.

Finally, finding the option to read the text was difficult: As he did not expect the function to be in the context menu, a user moved to the system tray to initiate the voice there. Only after a hint he returned back to KPDF and found the menu item. For the other participants, this did not cause a problem.

2.3.3 Reading via the Clipboard

Other KDE applications do not have a direct text-to-speech integration. Instead, users can select text in the application, copy it to the clipboard and then go to the Ktts manager and initiate the speech there. In the test, this interaction paradigm was tested in combination with the screen magnifier and for the KDE Mail application, KMail.

Each of the participants expected to find the function in the KMail menu or context menu. Only after a hint of the moderator they moved to the system tray recognised the Ktts icon or read the tooltip, and found the item "Speak Clipboard Contents" in the icon's context menu.



"Speak Clipboard Contents" function via the system tray.

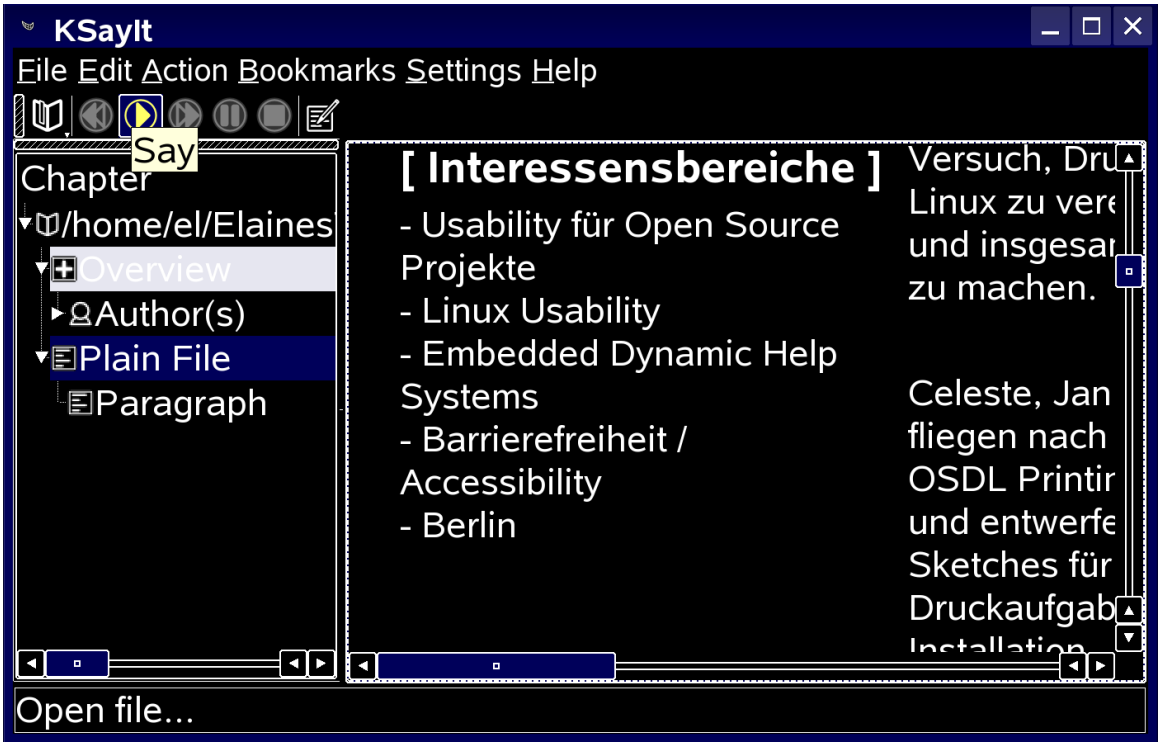
While the handling of the text selection was implemented in the default way and therefore easier than in KPDF, the frequently required navigation between mail application, system tray and manager window was evaluated as disturbing. The users were forced to re-orientate each time they chose another paragraph or wanted to stop the reading which distracted them from the actual contents that were read. The absence of global shortcuts to control the speech further complicated the usage.

Stopping the speech was required as the special character ">" that indicate comments in mails were read aloud by Ktts (see 2.3.7 "Quality of the Text Compilation") which was evaluated as disturbing and hindered the understanding

of the text quotes – especially in the case of second or higher level quotes.

2.3.4 Reading by Loading a File into an Extra Application

The application KSayIt allows to load text and html files and to have them read aloud. As an advantae compared to the three other types of text to speech integration, KSayIt provided controls to start, pause, stop the speech and to go to the next paragraph directly in the interface. Also, a navigation bar on the left that was supposed to mirror the document's structure should facilitate the navigation in the document.



The KSayIt window showing a html file, with navigation bar on the left and controls in the toolbar.

In the test it was shown that the given design had some disadvantages: First, when having a document read aloud, the text was displayed in the main window but the current position in the text was not marked. The users therefore could not follow the text, nor could they estimate their current position in the document. Second, images were not loaded into KSayIt. When the text referred to an image, the users had to pause the speech, open the original document, and search for the corresponding image in the text. As they had no idea about their current position in the text, it was difficult to find the corresponding section in the document. Third, the navigation bar on the left contained standard sections as “Overview”, “Author”, “Plain Text” and “Paragraph”. This structure did not apply

to any of the documents that were used in the test, and were therefore perceived as more distracting than helpful.

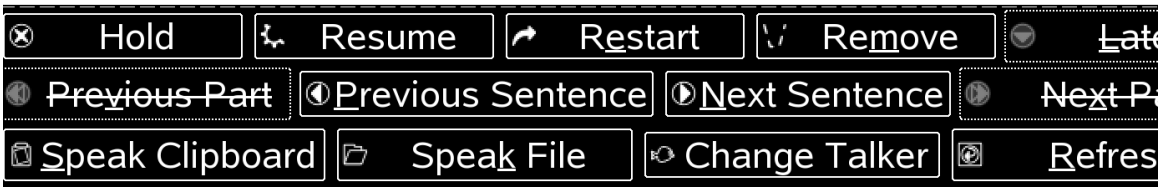
All in all the participants stated that they would prefer an integration of text to speech functionality with the applications instead of having an external application. In cases when an integration was not possible, they wished to import the complete document, including formatting and images, and wished the currently read part be highlighted in an eye-catching way.

2.3.5 Pausing, Repeating or Stopping the Reading

As mentioned above, the participants had to move to the manager window to control the speech. This caused a number of problems – starting from finding the window over locating the right controls to canceling jobs.

In applications with Ktts integration like Konqueror or KPDF, the participants partly expected the speech controls to be located within the application, for example beside the "Speak Text" item in Konqueror. As they could not find it here and already knew that Ktts was located in the system tray, they soon moved there and opened the manager window. Still, this re-orientation cost time and forced the participants to listen to text they tried to avoid, for example the language labels in Konqueror, or German text that was read in an English voice.

In the manager window, it was another challenge to locate the speech controls. Being faced with a complex tab layout, the participants had to identify the rather technical label "Jobs", and find the proper button in a block of thirteen visually ungrouped buttons.



The thirteen buttons to control speech in the Ktts manager window were mostly ungrouped.

As it was the first button below the job list, the participants mostly chose to "Hold" the current job. While the result was satisfying in the first run – the unfavored speech had stopped – "holding" had the effect that Ktts' job queue was blocked until the job was either resumed and finished, or removed. As a consequence, Ktts kept quiet when the participants wanted to have another piece of text spoken. Confused, they had to move to the system tray and the Ktts manager window and identify the source of the problem. Again, there were no shortcuts – neither to start nor to stop or remove a job.

2.3.6 Selecting Voices

In the test, one user had the problem that he wanted to have German text spoken, but there was only the English Festival voice installed. The participant therefore went to the "Talkers" tab in the manager dialog, and clicked the "Add voice" button. In the following selection dialog, he chose the German Hadifix voice and was confronted another dialog to further configure the voice.

In this dialog, the "OK" button was disabled, so the user was not able to save the new voice. While he later learned that the Hadifix voice was not installed on the system, Ktts did not provide the user with appropriate feedback why saving the voice was impossible.

For the quality of different voice packages and preferences among the participants, see 4 "Quality of Voice Packages".

2.3.7 Quality of the Text Compilation

Ktts provides simple text compilation (todo: what exactly?)

Still, the text compilation was not content-specific: Speaking an email needs to consider different text formatting styles than a PDF, a web page or HTML source code.

In the test, the fact that each content was handled equally by Ktts again and again caused problems: In an email, the special character ">" at the beginning of each line in quotes was read aloud which significantly hindered the understanding of the message. In an HTML source code editor, however, reading such characters might be preferred. In PDFs, headers and footers were thrown into the continuous text. On web pages, the approximately 50 items of a language drop-down were read.

While it may be difficult to find rules that apply to all cases in an application, the most common disturbances might be avoided by application- or content-specific text-to-speech profiles.

2.3.8 Integration with the KDE Desktop

In theory, each application that is run on the KDE desktop has text to speech support as its contents can be read aloud via the "Speak Clipboard Contents" functionality. In praxis, it turned out to be difficult for partially sighted users to frequently switch from an application to the system tray or Ktts manager. Repeating text they had not understood in the first run, pausing or stopping became inappropriately difficult by that means. Also, the participants missed an option to follow the text on the screen while it was spoken - none of them saw the "Current Sentence" text field in the manager window. Even then, pictures or graphics a text referred to could not be displayed there.

Given those difficulties, the current text to speech functionality is integrated with a few applications only. A KDE-wide framework that can easily be implemented in an application as an add-on is missing.

2.3.9 Conclusion Regarding Document Readers

Document readers are valuable tools not only for visually impaired users. The opportunity to have a document or long email read aloud instead of having to read it on the screen may be a facilitation to the majority of the computer users.

A document reader that fully supports these advantages needs to be easy to handle. Regarding the control and navigational scheme, a resemblance with music players would facilitate the initial understanding: Start, pause, stop and replay mechanisms, as well as options to jump to the next paragraph or chapter. Additionally to common music player controls, the speed of the voice and the language should changeable on the fly.

Instead of an external application, those controls should be integrated with the application to avoid a loss of the current task context, and they should be controllable by shortcuts. The currently spoken text should be highlighted in the application, and the viewport should scroll accordingly. An entry in the system tray would no longer be required.

The configuration of voices, notifications, and general settings should instead be available in a central place, for example the control center. By this, general settings can be applied to all applications offering text to speech integration. Still, application-specific configuration, for example reading “>” as “Quote level 1” should be located in an application's settings.

For non-KDE applications, however, there is still a need for an external text to speech application. It might combine the opportunity to read text from the clipboard, and to load complete files. Similar to a music player, it might hold a playlist.

Distributed text to speech controls, however, pose the challenge of finding appropriate priority mechanisms for jobs from different applications. When text is paused in one application, for example, initiating speech in another application should still work. System notifications should get a high priority, and possibly be able to interrupt another speech.

All in all, the user interface should be reduced to the main functionalities in the first run, to make sure visually impaired users can easily find relevant options.

3. Results: Support for Blind Users in Gnome

Regarding blind users, the Gnopernicus screen reader was tested. The participants performed the installation themselves, including the Braille support. Afterwards, they fine-tuned the system to fit their personal needs, and every-day tasks were performed while being observed and interviewed by the usability group.

3.1 Installation

The installation of Ubuntu Version xxx on a xxx was performed by two partially sighted and two blind users. The Ubuntu installation CD offered three different viewing modes for partially sighted users.

3.1.1 Installing the Ubuntu Base System

3.1.2 Gnopernicus Integration

dodo: Installation bla bla

After system startup, the users had to manually start Gnopernicus each time they logged in, it was not automatically started by the system. This also means that for the login, there was no screen reader support available. The participants explicitly mentioned this lack of support and wished to have a screen reader support right from the beginning.

3.1.3 Braille Support

3.2 Fine-Tuning Gnopernicus

After the base installation, the system was adjusted to the needs of the two users. As German native speakers, German language and its particularities had to be supported. Furthermore, the amount of audio feedback and notifications needed to be personalised.

To configure Gnopernicus, the users had to navigate to the Gnopernicus main window. It is the major location for adjusting Gnopernicus to the own needs and for trouble-shooting.

3.2.1 German Language

While the users had no problems to set the system and output language to German, Gnopernicus was not capable to handle the German "Umlaute" (Ä, Ö, Ü and ß) without further configurations. While they were properly displayed in the interface, the braille device left the letter out and the speech enumerated a row of special characters. The latter was perceived as disturbing.

After the base installation, the participants tried to change that problem, but were not able to find the right setting during the two days of testing. Todo: only in ui or also per command line? what did you try?

When interacting with the computer, the two participants handled the problem surprisingly well: In menus and other common interaction objects, they often assumed the meaning of a menu item (e.g. "Datei öffnen" - file open) and switched on to the next item or ignored the broken output.

Problematic was the understanding of text on web pages, where it was less easy to assume the meaning of words. Due to the broken "Umlaute", it was impossible for the participants to read German web pages or other documents.

3.2.2 Configuration of Audio Feedback and Notifications

Crucial for blind users is an appropriate audio feedback for actions and notifications. Almost as important as the feedback itself is an easy and accessible way to configure those options - otherwise they become "invisible" for the participants.

One of the most important types of audio feedback for actions is key press echo, which is handled in the first item "Speech" of the Gnopernicus "Preferences" panel. In some simple settings, punctuation style, the amount of text echo (letter, word or none) and the audio feedback when pressing modifier keys, navigation keys or spaces could be configured.

The style of feedback for notifications, however, could not be configured using the main panel.

3.3 Integration with the Gnome Desktop

Finally, the two users started to perform everyday tasks with the system in order to probe the Gnopernicus integration with the Gnome desktop. An important feature of Gnopernicus are different "layers" to navigate the desktop, allowing the user to reach interface elements and labels that can not be reached via tab sequences or keyboard shortcuts. By this, lacking accessibility support in applications is partly overridden.

A planned task, namely web browsing, was skipped because the German Firefox integration was not satisfying due to the broken “Umlaute”.

3.3.1 Layer Concept

Gnopernicus possesses a number of functions that are mapped to the numeric keypad. As there are more functions available than keys on the numeric keypad, Gnopernicus introduced the “layer concept”: Pressing 0 on the numpad, quickly followed by another number that represents a layer, will make the system switch into that “layer”, and the keys on the numeric pad represent the new layer scheme.

Regarding navigation on the screen, the most important schemes are the focus tracking mode and the flat review mode of Layer 0. The user can switch between these two modes by pressing DEL on the numeric pad. While the focus tracking mode allows for a quick navigation to static interface elements like the toolbar, menu or previous line, the flat review mode provides an opportunity to read the screen like an image. With the help of the numeric keypad, the user then navigates up, down, left and right on the screen, while interface elements he passes are read aloud.

In the test, the flat review mode turned out to be an important tool to reach interface elements that were not accessible via the keyboard. One such element were HTML descriptions in the tool to add and remove software, another element was the chat window in Gaim, an instant messenger, which was not read aloud in normal mode.

3.3.2 Performing Common Tasks

Having set up the system to mostly fit their needs, the participants were asked to perform several everyday tasks like file browsing, burning a CD or reading a PDF. Due to time restrictions, each task was performed by one of the two blind participants only.

During the test, it happened several times that Gnopernicus stopped to speak. The reasons were mostly unclear to the users as well as the observers, so the users used “Alt+Tab” to search the screen for windows that might block the speech. When they did not find the reason for the problem, they mostly restarted the system - even if the Braille device was still capable to read the screen. They claimed that speech would be better and easier to use, so they did not want to miss it.

3.3.2.1 File Browsing

First, one of the participants was asked to browse the contents of a CD. As the

CD was mounted automatically by the Gnome file browser Nautilus, reaching the CD did not pose a problem to the user. But also navigating freely between folder hierarchies was easily accomplished by him.

In a next task, the user was asked to copy all contents from a USB stick into a folder in his home directory. While most of this task was also accomplished without problems, the user was faced a problem when he wanted to navigate to the home folder using tab, but nothing happened as all elements were still selected. Only after deselecting them, the tab navigation worked again.

When navigating the elements in a folder, the user used both, right/left and up/down keys. As the default Nautilus view was Icon view which displays elements in rows and columns, using both left/right and up/down keys for navigation was appropriate. However, this navigational scheme had some shortcomings: When the user reached the lower right element, he never knew if it was the last element in the list as the down key did not move him further down. However, using the right key showed him that there was an additional item. A more flexible keyboard navigation as well as an announcement of the overall number of elements in a folder would be helpful here.

All in all, the user missed an announcement of access permissions when going through the list of elements. During the test, it happened that the user wanted to open a text document which was locked for him. As a matter of fact, he was not allowed to open the document, but Gnopernicus did not tell him about the access permissions.

3.3.2.2 Burning a CD

When burning a CD, it could clearly be seen that the participant was not a Gnome beginner: Instead of searching for a tool in the main menu, he launched the "Run command" dialog and entered "nautilus-cd-burner", the name of the file browser's burning tool.

When starting the burning tool by that means, a modal dialog popped up telling the user that there was no file selected. He then had the choice to either close the application or open the CD/DVD creator. Neither the informational text in this dialog nor the buttons were read by Gnopernicus, it kept silent. This was astonishing, as all elements of the dialog could be reached by the keyboard, as the observers stated. Still, the user could not proceed without the help of the moderators.

In order to add files to the CD, the user simply went back to the Nautilus window that was still open from the last task, copied the contents he wanted to burn on CD, and inserted them in the CD application. Finding the item to start the writing in the "File" was not a problem, and a window titled "writing file to disk" popped up. However, there was no automatic feedback informing the user about the write progress.

When the CD was finished, Ubuntu's auto-mount function made a Nautilus

window to pop up, showing the contents of the CD. By browsing the contents of the window, the participant could evaluate the success of the writing.

As the CD burner itself did not provide a system-wide feedback, it is questionable if in case of a problem during the burning, the user would have realised it.

3.3.2.3 Reading a PDF

The support for reading the contents of a PDF is generally a problem - a reason, why PDF is an unfavored format among blind users. Current initiatives like accessible PDFs (e.g. by OpenOffice.org) aim at changing that situation, but the majority is still inaccessible for blind users.

Also in the test, reading a PDF by means of the Evince PDF reader (Gnome default) was not possible. While the text in the document was completely invisible to Gnopernicus, the table of contents informed the user that there actually were contents ("Table, 1 symbol, 2 symbols"), but could not read the contents themselves.

The contents of this PDF, created by OpenOffice.org, could neither be read by help of the focus navigation, nor the flat review mode which usually allows blind users to reach interface elements that cannot be reached by other means.

A workaround one of the participants reported was to create a Linux shell script that converted PDFs to text, and afterwards converted that text to MP3. As the original text usually contained headers and footers for each page which was disturbing when listening to the text, he made use of flexible replacement tools.

3.3.2.4 Writing Text

In a next task, a participant was asked to write text with help of the editor Gedit. The user was not faced any problems here: As he had set the speech options to echo each word before, he got immediate feedback about the correctness of what he typed.

In Gnome, modified but unsaved documents are marked by a star next to the window title. When asking the participant about the meaning of that star, he guessed it would mark the currently active window. The actual meaning, "modified", did not get clear to him in the first run.

3.3.2.5 Reading Mail

The task of reading mail posed one of the greatest challenges to the user. First, he tried to make use of Evolution - but the application kept crashing when using it in combination with Gnopernicus.

Therefore, the user and observers decided to install Thunderbird, which was said

to have a better accessibility support. After the installation, the participant tried to set up his email account, but the text input fields in the Add Account wizard did not possess any labels. He could only guess which values to enter into which fields, and decided to have somebody set up the account for him.

When the account was set up by one of the observers, the participant went back to Thunderbird. While he was able to navigate through the messages, the user felt lost and could not determine the subject of the mails. It was unclear to him where he was, where to find the message body, and finally gave up.

The other user, who had already tried Thunderbird before, explained that Gnopernicus would not be able to read Thunderbird's mail index which would significantly hinder the understanding of the navigational scheme.

3.3.2.6 Chatting with a Buddy

Finally, one participant was asked to set up his instant messenger and chat with a friend. The user started Gaim, the Gnome instant messenger application.

Directly after launching the application, a wizard to add an account popped up. While adding an account was not much of a problem, the user was surprised when he chatted with a friend, but got no answers. Actually, they appeared on the screen in the chat window, but the contents of the window were not read. He got audio feedback on new incoming messages, however, and also the words he typed were echoed.

When he was informed by the observers that the friend's messages were actually displayed on the screen, the user switched into flat review mode. He moved over the window, and finally found the text his friend had written. To answer, he had to switch back to focus navigation mode, then back to flat review to read the response. As a matter of fact, this way of communicating was perceived as very exhausting.

3.3.2.7 Installing Applications

One of the most problematic fields was the installation of new software. When Evolution kept crashing, it was decided to install Firefox, which offered a realistic setting to probe software installation with Gnopernicus screen reader support.

The user chose the “Add Software” item in the Gnome “Applications” menu which opens the “Gnome App Install”. In this tool, software categories were displayed on the left, and the available software was displayed in a HTML container on the right, listing applications names and short descriptions.

As the HTML container could not be read in the focus navigation mode, the user had to switch to the flat review mode. By this means, he could read the application names. As he did not want to read through the whole list, he started to type “Mozilla” (for “Mozilla Thunderbird), but the cursor was in the wrong list

and did not find a result. His first thought was that Thunderbird was not available.

After a hint by the moderators, he looked for a search field to enter the application name. Back in focus navigation mode, he found a field labelled "Textfield", and assumed it might be "Search". He entered "Mozilla". As the label "Mozilla" was not part of the description, the search list remained empty. Given this kind of feedback, the user could not know if there actually were no results, or if Gnopernicus was not able to read them. In order to probe it, he entered Nautilus, found a result in the flat review mode, and finally entered "Mail" into the search field.

In the following steps, the software package was downloaded and installed. Both feedback was not visible to Gnopernicus, so the user could only assume that the installation was successful. All in all, the user was faced lacking feedback and accessibility support throughout the installation routine.

3.3.3 Advantages of a Graphical User Interface

After the test, the two participants were asked for an evaluation of their experiences with the Gnopernicus screen reader. While the usage was relatively new to Sebastian, Henning had made use of it for a year now.

Sebastian valued the progress the screen reader had made during the last year. Even if he was faced problems and missing Gnopernicus support during the tests, he found that the usability and opportunity to access applications had much improved. Even if he claimed that he would not yet use it in a work environment, he liked to experiment with it and probe the functionality.

Henning, as a frequent Gnopernicus user, valued the underlying technology which that tries to keep load away from the screen reader, but makes applications compile preprocess the data which is given to the screen reader. Due to that technology, he claimed, it would take more time to make all applications accessible. Due to this lack of consistent support he used Gnome in combination with the console.

To them, the advantages of a graphical user interface compared to a console were apparent: Being able to read the same document formats as the majority of the computer users, burning CDs with the advantages of a graphical user interface, or ... todo

4 Quality of Voice Packages

4.1 German Languages

4.2 American English Languages