Presenting an extensive list of settings to the user

Sjoerd Langkemper
stlangke@cs.vu.nl

January 21, 2005

Abstract

This paper addresses some problems which show themselves when a extensive list of settings or options is shown to the user of a program. It lists some examples and tries to identify the problems in those examples. Furthermore, this paper tries to suggest some recommendations to keep in mind when developing the 'settings' part of a program.

1 Introduction

Many modern programs are configurable. This means that the user can alter their behaviour during run-time. This can be done in a number of ways, such as editing a configuration file or by a dialog box. In a word processor, for example, the user can set the default font, enable or disable the spelling checker, or disable that annoying paper clip.

If a program has many settings available, it gets difficult to present them to the user in a orderly way. This paper will address some of the issues with designing such settings dialogs.

2 Why configure?

Configuring a program is done to adapt it to its environment. For example, a browser may need to be told where the proxy server is. The proxy server is a piece of the environment and if the browser does not know about it, it will simply not work.

In the good old days, you had to configure your hardware too. Setting jumpers to indicate the interrupt line and I/O address were not too uncommon. Subsequently, you had to tell the software what you had set so the software knew how to find the hardware.

This has become largely automated. With PCI and "Plug and Play" devices the computer and the device can negotiate over settings and the software can detect what kind of hardware it is and where it is located.

The proxy settings in the above example have also been largely automated. If the system administrator provides the settings on one server, all clients can automatically adopt these settings.

Most configuration can be automated. With this in mind, autoconf[6] was developed. It's task is to configure several host-specific options of a program just before compiling it. For example, is this a 32-bit or a 64-bit computer? Users of 64-bit computers now do not have to indicate this, the autoconf program figures it out by itself. This is a big advantage, because the user may not even know what kind of computer she has. With automatic configuration, she does not need to know.

There are two reasons why configuration can not be automated fully. First, for performance reasons. Finding a suitable proxy server can be done by connecting to every IP address on the network. This would take a very long time and it is much easier to ask the user.

Second, there is one factor which the software can not automatically determine: the will of the user. The program can not by itself decide whether the user wants a blue or purple background. Configuration can thus never be fully automated. However, if it can be automated, it should.

3 Examples of setting dialogs

3.1 Linux kernel

The Linux kernel has a lot of settings. As you can see in fig. 1, the options are categorised for a better overview. Most options can be either enabled, disabled or put in a module.[2]

Furthermore, it lets the user choose what the layout of the menu should be. When *Single* is selected, there is only one screen and the contents of the submenu appear in this screen. With *Split*, the menu tree is on the left of the screen and the settings are displayed on the right side. *Full*, which is the default, puts all the settings in one big tree.

It is pretty nice to let the user choose the layout of the settings. However, users may not choose the most efficient layout for them.

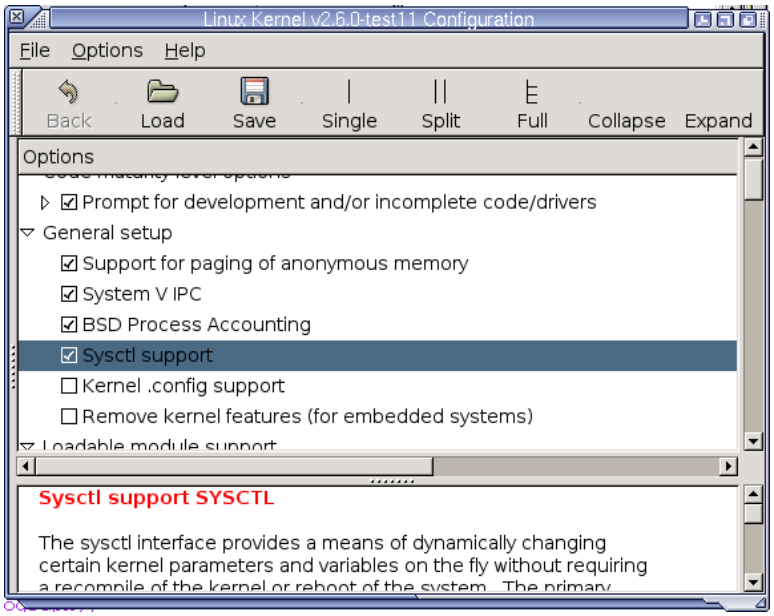

Figure 1: Linux kernel configuration

3.2 Mozilla Firebird

This screenshot from the Mozilla Firebird[1] web browser shows the preferences dialog. When the user selects a category on the left, the menu of right side of this dialog changes. The form on the right is not always the same, but it's layout changes to best represent the settings available in this category.

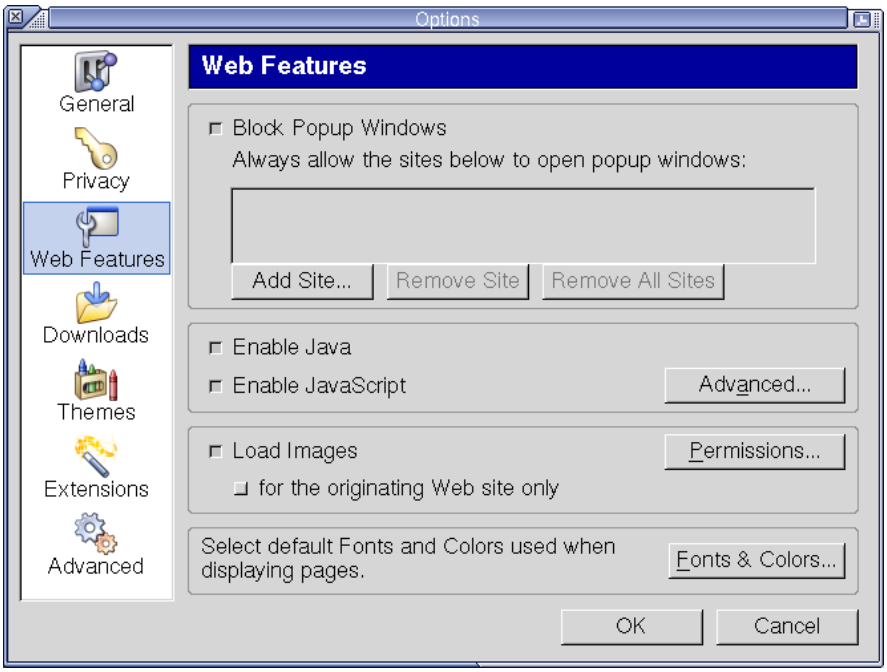

Figure 2: Mozilla Firebird configuration

3.3 Windows 2000 network settings

The screenshot above is from Windows 2000. This are the dialogs with which one can set the network settings, such as the protocol and IP-address to use. It makes use of tabs and some buttons open a new dialog.

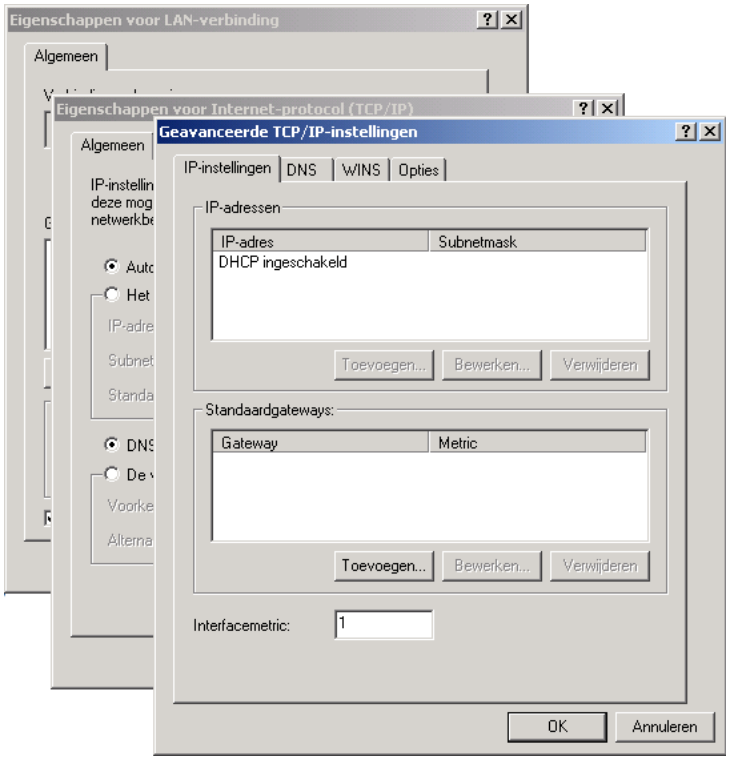

Figure 3: Windows 2000 network configuration

4 Possible problems

- screen real estate - all the settings should be easily accessible, although they may not fit on the screen;
- information hiding - the settings intended for expert users should be hidden or hard to access;
- dependencies between settings - when the spell checker is disabled, the language settings for the spell checker should also be disabled;
- linearity - when the user first uses the program, she might want to set all options at once;
- searchability - the user should be able to find a specific option;
- updatability - when the settings are changed the user may want to set all the new options.

4.1 Screen real estate

This is of course the most basic problem when dealing with many settings. If all the settings would fit on the screen, there would be no trouble at all. With screens getting bigger, one might think that this problem will solve itself. However, putting your new 21" display full of settings does not make it easier to use.

4.2 Information hiding

Not all users are the same. The program does not know whether the user already knows all about the program, or that she is just starting with it. To minimize mistakes, dangerous settings and settings that are seldomly used should be hard to change or hard to access.

For example, a dishwasher can heat the water to any level. There could be a turnknob on the front of the dishwasher to indicate the water temperature, from anywhere from zero to 70 degrees centigrade. However, the user just wants the dishes to be clean. She does not have to know what temperature is used when. Therefore, a better design would be a dishwasher with one or two buttons: clean very dirty dishes (60°) and clean not so dirty dishes (50°).

4.3 Dependencies between settings

One setting could exclude another. For example, one can not enable TCP/IP in the Linux kernel, when networking support is disabled. In most programs, these dependencies stay to a minimum, or they are really simple for the user to understand: one can not set the expiry time on cookies when cookies are disabled.

Dependencies are easy to understand if these settings are close to another, preferably in the same group. It gets really hard for the user when a depending setting is not close to the setting it depends on. This can be by bad design, but sometimes its just not possible or feasible to place items according to dependency. Now another method is needed to describe the dependencies of a

setting. Is this not done, then the user does not know why that particular setting was there the last time, or why it did show up at his coworkers computer.

4.4 Linearity

Sometimes the user wants to view and possibly change all settings one by one. This is especially true for the Linux kernel, but also for almost every other program: a user might want to view all the settings to make sure everything is to her liking, or just to see what can and what can not be set.

The program therefore should offer a way so that the user can browse the settings one by one.

4.5 Searchability

Sometimes a user wants to change one option. For example, a user has gotten another irritating advertisement pop-up and she decides to disable pop-ups. Now the user wants to find the checkbox which says *Block Pop-up Windows* as soon as possible.

Searchability is probably the most specific property of extensive lists of settings. Normal interfaces have a more linear form, where the user has to fill in all text fields, for example. Although this is also the case with settings, occasionally the user wants to change a specific item.

4.6 Updatability

When the list of settings is changed, for example by an update of the program, the user might want to look into the new settings. Just as a user would scroll through the settings when she gets a new program, she would look at the new settings when she gets a new version.

5 Improvements and solutions

5.1 Screen real estate

When not all settings fit on the screen, the program has to show some at a time. To solve this problem, there are basically two solutions:

Make the screen (artificially) bigger - Of course a software designer can not ask his customer to buy a bigger screen. He can, however, make better use of their screen, by pretending it is larger. The user can then scroll to the part they want to see. This is often implemented with scroll bars, although in some situations movement of the screen is done by moving the mouse pointer to the edge of the screen.

Divide the screen in multiple parts - The list of settings can be split up to form sets of settings. Settings which have a relation to each other are often in the same set. A set is displayed when the user indicates that she wants to see one.

The second solution is preferred and more common, because grouping settings has some more advantages, as is shown below.

5.2 Structured display

When the user goes over a number of settings to change these to her liking, she has to think about what value these settings should have. Does she want the spell checker to be enabled? Does she want a white or a blue background?

The settings should be logically grouped for two reasons. First, so that the user can decide what she wants on a particular *subject* instead of a particular *setting*. For example, if the settings about colour begin, the user starts to think how her word processor should look like. For example, she chooses red text on a blue background, with green line numbers. If these settings are not close to another but randomly placed, she has to think about the colours three times instead of one time.

The grouping of items is also important because it is the only hint to where a setting may be placed. It is probably a good idea to list all the settings and place them in groups accordingly. Keep in mind that the number of settings in a group should be about the same for different groups, so that the number of settings on a dialog do not vary too much[9]. Furthermore, groups like *General*, *Other* or *Advanced* to hold items which do not fit in any of the other groups are generally not a good idea.

Most programs have grouped their settings logically. The interface then consist of two parts, with in one part the list of groups and in the other part the list of settings in that group. Mozilla Firebird uses a list of icons to indicate groups. In Microsoft Windows, the list is in the form of tabs on the top of the dialog. When a group is selected in the list, the settings which belong to that group are displayed.

5.3 Information hiding

Information hiding is commonly used in programs. Most programs have so much features and settings that is would be hard to put all these settings in an interface. However, expert users stop at nothing to let the program do what they want.

As you can see in the example of the Mozilla Firebird browser (fig. 2), it has a button *Advanced*, which opens up a dialog with expert settings.

If you are a real expert, you can type `about:config` in the location bar of Mozilla Firebird (fig. 4). A list with settings pops up which the user can change. There are no pretty interfaces for most settings, the user just has to type a value in the box corresponding to a particular setting. This is not an easy to use interface, but it is not meant to be. This section is just for the people who want to change the behaviour of the program at all costs.

The configuration of the Linux kernel shows only advanced options when "Prompt for development and/or incomplete code/drivers" is set. Even when this is set, such drivers are marked as experimental.

Windows 2000 Network Settings has buttons like *Properties*, *Advanced* and *Settings* on almost every tab, with fairly normal settings behind them. This hides the expert options pretty good, but makes it a bad interface for more

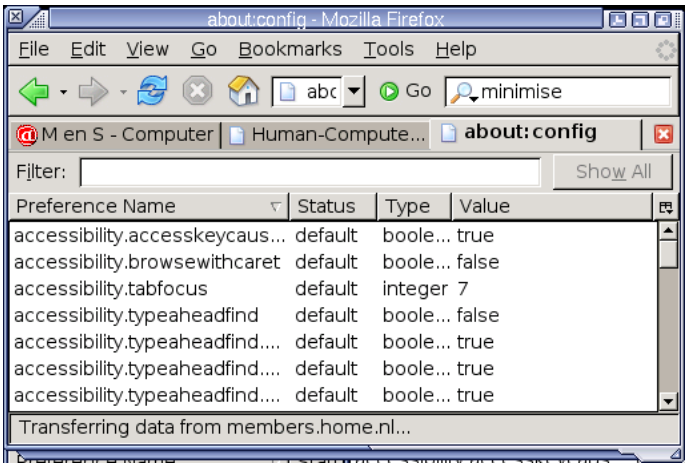

Figure 4: Mozilla Firefox' about:config

experienced people, as they have to work their way through five dialogs just to get to the one they want.

5.4 Dependencies between settings

In the Linux kernel example, some settings have their own list of options below them, slightly indented. These depend on another, and the list is deleted when the dependency is not longer met.

Many settings depend on the setting described above, "Prompt for development and/or incomplete code/drivers". These dependencies are not clear, but this option is only for expert users who know how to enable and disable incomplete drivers.

As you can see in fig. 3, no IP addresses can be added. The short text "DHCP enabled" describes that this field can not be used because the computer gets an IP address automatically. It could be more elaborate, but at least there is some explaination to why the buttons are disabled.

5.5 Linearity

A feature should be in the interface of the configuration that lets the user scroll through the settings one by one. Adobe Photoshop[4] has handled this nicely, by putting *Next* and *Previous* buttons on the settings dialog (fig. 6).

The text interface of the Linux kernel configuration provides linearity by asking one setting at a time, but you lose the nice graphical interface. Furthermore, it has its own problems as described in section 7.2.

5.6 Searchability

Settings are often grouped to facilitate this. A specific setting is found in a particular group. When the user thinks of a specific setting, it decides in

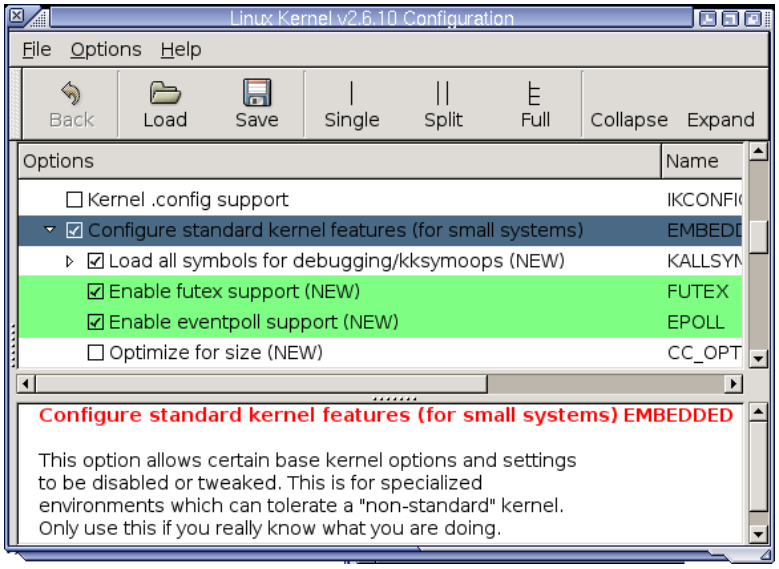

Figure 5: Dependencies in the Linux Kernel configuration
Green items are hidden when the blue item is not enabled.

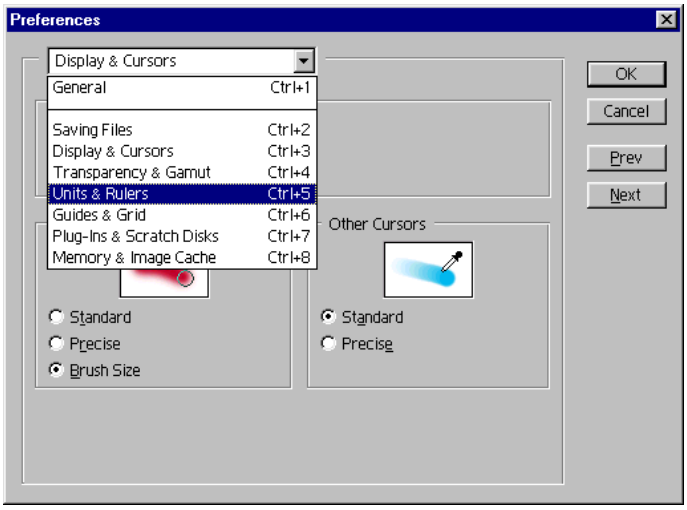

Figure 6: Linearity in the configuration of Adobe Photoshop.

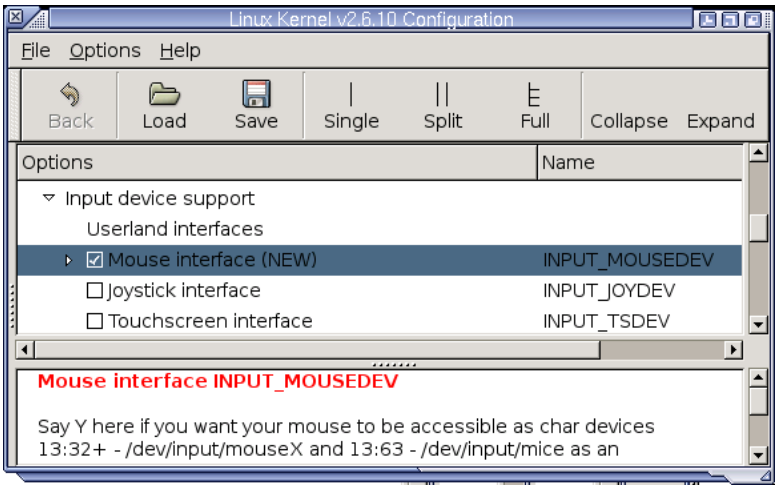

Figure 7: Linux kernel configuration - indication of new option

which groups it fits best and then searches in that specific group. This is done repeatedly in tree-like layouts.

For a setting layout to be searchable, it is thus very important that the groups are carefully chosen.

Mozilla Firefox' `about:config` as shown in fig. 4 has a real search bar, but you still would have to know the name of the setting to use it efficiently.

Some parts of Windows Network settings are accessible by more ways than one. This effectively places some settings in two or more groups, which can enhance searchability. It is important that the user sees that the two ways lead to the same setting dialog.

5.7 Updatability

Updatability is not one of the most important issues. In some programs, updates change so much of the program that the whole configuration interface is also changed. In this case, the user can step trough the settings linearly.

In other cases, updates are minor and new settings are added without changing the interface. The configuration of the Linux Kernel is an example of this. New features are indicated as such in the graphical interface (see fig. 7).

Furthermore, in the text interface of the configuration, the user has the option to only configure the new settings. If the user has previously configured the kernel and updates, the configuration interface asks only what the user wants on the new settings instead of walking through all settings again.

5.8 Tabs

In Windows, almost all configuration interfaces consist of tabs which are supposed to split the settings in groups. However, when there are so many tabs in an application that they take more than one line, overview is lost completely. To be able to read the text on the tabs quickly, they have to be aligned. When

all the tabs fit on one line, they are all aligned from left to right and there is no problem. When there are multiple lines, the tabs are placed in a haphazard way, not aligned at all. [9] When the user clicks on one of the tabs, they are ordered differently. This is not only confusing, in makes browsing through the tabs in a linear way impossible.

6 Reflection

Now that we have seen all issues with designing configuration interfaces, lets see how the programs in the examples handle these things:

	Mozilla Firefox	**Windows Network**	**Linux kernel**
Layout	Grouped settings.	Tree with dialogs and tabs.	Tree with boolean settings.
Screen usage	Icons with settings (same dialog).	Tabs and new dialogs.	Scroll bar.
Information hiding	'Advanced' group and buttons.	'Advanced' buttons (several layers).	Setting to enable 'dangerous' options.
Dependencies	Indention and closeness.	Textual and closeness.	Indention.
Linearity	Fair, walk each group.	Poor, walk complicated tree.	Fair, walk tree from top to bottom.
Searchability	Fair, by group.	Poor, experience and guessing.	Fair, by group.
Updatability	None.	None.	Fair, indicated as new.
Structured display	Fair.	Fair.	Good.

The Linux kernel has the most settings available of the example programs, but the easiest interface. This could have something to do with that most settings are boolean: no fancy dialogs are needed to configure each option.

The Mozilla Firebird configuration interface could be easy, if the groups were better thought out. For example, "Block Popup Windows" could be under the groups "General", "Privacy", "Web Features", "Extensions" and "Advanced", which are five out of seven groups.

The Windows Network setting do a great job at information hiding, but are harder to use by expert users as a result. If you do not know where a specific setting is, it is very hard to find.

7 Other interfaces

Not all interfaces to settings are the same. For applications with other properties, for example those without a graphical user interface, other interfaces to settings may be a good idea.

7.1 Command language

In a command language, the user needs to know how to use every keyword. For an extensive list of settings, it may not be the best interface, since the user

has to memorise pretty much. The searchability and linearity are bad. The hiding of options is well done, but for all users instead of only experts.

A command language is often a powerful tool, because commands or queries can be used to tell the program exactly what to do. For settings, however, all the computer needs to know is what value a particular variable will get. The powerful command language is not used to it's limits.[8]

Although it is generally not a good idea to use command language to change settings, some programs might have no choice. Many simple network servers, for example, are only accessible by a terminal client (telnet), which is pretty basic on some systems. Other programs have chosen a command language interface for the whole program. Settings should also be in command language, to keep consistency. A good example of the last is Mutella[5], a peer-to-peer network client. Although program also has a web interface, the settings are only accessible through the command language interface[1].

In Mutella, the settings are grouped. This improves searchability. Linearity can be obtained by editing a configuration file by hand. Mutella also offers the searching of settings on keywords. For these last two things, one has to be an expert user.

7.2 Question and answer

Question and answer interfaces are inefficient, but are self-explanatory and simple. The old style configuration of the Linux kernel uses a Q&A interface, for people without a graphical interface. The linearity is high, but settings are not searchable and the order in which the settings are handled is fixed. Another important disadvantage of a Q&A interface is the lack of forward context. For example, the Linux kernel first asks the users if she wants to include something called the *Kernel automounter (version 3)*. Only then it asks whether she wants to include the *Kernel automounter (version 4)*, which supports version 3, as well. [8][2]

8 Conclusion

As described above, there are a number of properties important when designing a interface to an extensive list of settings:

- automation - configuration should be as much automated as possible;
- screen real estate - all the settings should be easily accessible, although they may not fit on the screen;
- structured display - settings should be divided in groups in a clear and unambiguous matter;
- information hiding - the settings intended for expert users should be hidden or hard to access;
- dependencies between settings - dependencies between settings should be made clear; the language settings for the spell checker should also be disabled;

[1] Newer versions now also have settings accessible through the webinterface.

- linearity - it should be possible to scroll through the settings one by one; all options at once;
- searchability - it should be easy to find a particular setting;
- updatability - it should be clear what settings are new after an update.

Most interfaces to settings are the same, namely a list of groups, each associated with a list of settings. In this type of interface, it is most important to group the settings in a clear, unambiguous manner.

References

[1] Mozilla project - http://www.mozilla.org/

[2] Linux kernel - http://www.kernel.org/

[3] Windows 2000 Professional NL - http://www.microsoft.com/

[4] Adobe Photoshop 6.0 EN - http://www.adobe.com/

[5] Mutella - http://mutella.sourceforge.net/

[6] Autoconf - http://www.gnu.org/software/autoconf/

[7] Human Computer Interaction - Alan Dix et.al. - 1997

[8] Principles and Guidelines in Software User Interface Design - Deborah J. Mayhew - 1992

[9] Designing Visual Interfaces - Communication Oriented Technique - Mullet & Sano - 1995

List of Figures