Improving Usability of the Linux Kernel Configuration Tools

Kacper Bak
Generative Software Development Lab
University of Waterloo, Canada
kbak@gsd.uwaterloo.ca

Karim Ali
PLG Group
University of Waterloo, Canada
karim@uwaterloo.ca

ABSTRACT
Tailoring a Linux kernel to one's needs has been one of the most cumbersome tasks a GNU/Linux user can do. There have been many attempts to overcome this problem by introducing smarter configuration tools. Those tools, however, still lack some important features, which discourages users from using them. In this project, we addressed the problem of usability of the Linux kernel configuration tools. We identified the major usability issues with current tools, proposed a better user interface and evaluated it on a group of Linux enthusiasts.

Author Keywords
usability, linux, kernel, configuration

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: Miscellaneous

INTRODUCTION
GNU/Linux is a free operating system with Linux as its kernel. The Linux kernel[1] is a mature and very complex piece of software. It has been developed by thousands of programmers led by Linus Torvalds since 1991. The kernel supports a variety of computer architectures, network protocols, thousands of drivers, and many debugging options.

Structure of the kernel is very modular, so that each user can tailor it to her particular needs and specific hardware. Recent research [16] has shown that the whole kernel is composed of almost 5500 features, of which 89% are user-selectable. Thus, the variation space is huge and the configuration process requires a broad computer knowledge. Users can configure kernel either by manually editing the .config file or by using some configuration tool (i.e. menuconfig, xconfig, gconfig). The first option is discouraged, because it is very error-prone due to lack of automated constraint validation/propagation. Using a configurator is a better solution, but it is still a laborious process.

[1]Available at: `http://www.kernel.org/`

Existing configuration tools have been evolving for over a decade, but the progress is reasonably slow compared to development of other parts of the kernel. Usability seems to have low priority on developers' task lists. To address this issue, we started a course project aimed at making the configuration process easier and more intuitive for Linux enthusiasts. In the project we investigated current configuration tools, identified major usability issues and proposed alternative designs. The designs were then presented to a group of users who provided comments and constructive feedback. Finally, we chose the most promising design, implemented it and evaluated. General reactions of the test group were positive and participants said the new tool was an improvement over the standard tools. We believe that our findings will help in creating better configuration applications not only for the Linux kernel, but also for other kinds of variability models.

The paper is organized as follows. We describe the problem domain and results of our initial study of standard tools in Sect. . We explain how we reached the new design and how it compares to xconfig in Sect. . We then present evaluation of the new tool in Sect. . We discuss future work in Sect. . After presenting related work in Sect. , we conclude in Sect. .

PROBLEM
People configure operating system kernels for several reasons. Usually they want to customize kernels to specific hardware or to particular needs. This scenario is very common in embedded systems, such as network routers, where hardware resources are very limited. By carefully selecting options, one can improve system stability or functionality. Although many configuration options are related with hardware and device drivers, there is also a large group of options that change the way operating system works. A good example is "preemptible scheduler" that makes Linux a soft real-time operating system. Such a scheduler is desirable for digital signal processing, e.g. for recording and transforming signal from the guitar.

Majority of users configure kernels without even realizing this fact. We distinguish two types of kernel configuration: static and dynamic. In the former method software is customized before compilation, so that only chosen pieces of code are compiled. It results in a smaller program footprint and faster compilation. On the other hand, later reconfiguration requires more effort, because any additional piece of software has to be compiled separately. In many modern dis-

Submitted for CS 889 - Open Source Usability.

tributions users are provided with fully functional kernels, that do not require further compilation. Instead, they dynamically customize the software, by loading relevant modules. The disadvantage of the second method is that preparing such a big kernel takes a lot of time and resources. Furthermore, it is infeasible to apply it to embedded systems.

Regardless of configuration type, there is still need to customize the software. Although this task can be done manually, users prefer to use automated and intuitive tools, that guarantee system correctness. Unfortunately, the software currently used for kernel configuration is neither fully automated, nor easy to use. In addition, static and dynamic configuration are two separate mechanisms and are supported by completely different programs. Static configuration is done by the Linux Kernel Configurator, while dynamic configuration is about adding or removing compiled modules. In our opinion the two worlds can be merged by providing a single user interface.

Linux kernel configurators are mostly targeted to hackers. Some kernel developers expressed opinion [13] that kernel should be configured only by experienced users. While this statement might be true, there are still lots of Linux novice users who want to learn how to configure kernel or need to compile/load a particular driver. As of now, there are millions of web pages describing how to configure and compile the Linux kernel. Novice users are often overwhelmed by the number of available options and required knowledge. To understand challenges of configuration, we carried out an initial study on a group of Linux users.

In the study six participants were asked to statically configure the Linux kernel for a popular laptop for typical home or office use. They used the standard xconfig tool that comes with the Linux kernel. It presented a list of options in a tree structure composed of categories and options. A person succeeded if the new kernel was ready to play music and movies, connect to Ethernet, use WiFi, memory cards, USB devices, Bluetooth, CD/DVD. This scenario seemed rather common, because in many Linux distributions users have to fine-tune their configurations to make the whole system work as expected. None of the subjects succeeded with the task without further help. We observed many problems that people faced during the configuration process. It is worth noting, that our participants had previous experience with Linux and claimed to be advanced computer users. During the study, we were also interested in how they used xconfig, and how people configure software in general. Our findings are summarized in several categories:

Menu hierarchy All participants had problems with selecting drivers for laptop's hardware. First of all, the tool showed the whole menu hierarchy, and people said they were overwhelmed by thousands of options. Several of them started with collapsing all menus, so that only top-level categories were visible. Furthermore, the tree/subtree relationship was confusing. It was unclear whether selection of parent option implies selection of required functionality or additional suboptions should be selected as well. To some, even such a basic thing as checking an option was confusing. Besides familiar empty and filled checkbox, the tool showed a box with dot inside. It meant that the option will be compiled, but as a dynamically loadable module.

Option names and descriptions Cryptic names were another source of problems. The existing infrastructure uses option names that make sense to programmers and kernel hackers, but are hard to understand for less experienced users. Unfortunately, descriptions were not very helpful, since they often explained implementation details instead of end-user functionality. Participants expected to see what an option means to them and whether the device driver is appropriate for their hardware. In many cases their expectations turned out to be too high.

Searching Huge hierarchy of options and cryptic names make it very hard to find a particular feature. All participants used search in xconfig. Searching worked very poorly, since feature descriptions were skipped by the tool. People expressed their frustration and often used Google to match modules with hardware, and later select the module in xconfig. Google was also used to find a list of drivers required by the laptop. For all users, querying a web search engine was easier than running system tools to discover hardware devices.

Configuration We carefully observed how users used current tools to configure a large piece of software. Most of them were jumping between different categories as they tried to locate modules. One person, who had experience with kernel configuration tools, started menuconfig program, which performs wizard-like configuration. The problem with this tool was that the user could not jump to other categories, but had to answer many irrelevant questions. After a while, the participant switched to xconfig. Users had problems with selecting appropriate device drivers. When they were unsure about which module to choose, they selected all the modules that seemed relevant.

The initial study was a necessary step to discover user expectations and needs. It led us to the following conclusions:

1. Menu hierarchy should be as simple as possible. There are far too many options that could be reduced automatically if the configuration tool targeted specific user group and had good reasoning capabilities.

2. Feature names and descriptions should reflect end-user functionality instead of low-level details.

3. Powerful searching capabilities are crucial as the number of configuration options grows.

4. Automatic hardware detection and kernel autoconfiguration is important, since majority of kernel options is related to hardware drivers.

The aforementioned problems can be fixed by improving application's backend and creating a more intuitive user inter-

face. Our project focused on constructing a GUI that would be intuitive, simple and targeted to specific group of users.

LINUX KERNEL CONFIGURATION TOOL

To design such configuration tool, we first set our target group to be less experienced Linux users. In other words, we are building this tool for novice and intermediate Linux users to allow them to to configure a Linux kernel withouth burdening them with many of the unncessary details involved during such process. We started the design process of our Linux Kernel Configuration Tool (lkc tool) by designing several mockups that have our vision for the tool. Initially, we thought that once the tool is started, it should first fetch the current Linux kernel configuration, then detect the currently connected hardware. A dialog box (splash screen) should be shown to the user at that time to show the progress of these processes. Figure 1 shows four different mockups for that dialog box.

We decided to do a second user study that included six participants. The aim of this study was to show the participants those designs and get feedback from them regarding what aspects in those designs were appealing to them, and what were not. That study led us to draw the following conclusions:

1. Users always liked to know what is going on. In other words, they did care about getting more information about what the tool is doing while the progress bars are being filled up.
2. The fact that users appreciate access to more information does not mean they appreciate that such information should be displayed by default, only when requested.
3. It is highly desirable to have the tool report to them any crashes that occur during the hardware detection process.

Consequently, we came up with a design and implemented a prototype for our splash screen as shown in Figure 2. In the prototype of the splash screen, the user will be asked a question whether she wants to detect the current hardware or not. If she answered yes to that question, she will be presented with the dialog shown in Figure 2a. The list of tasks along with the progress bar gives the user an idea about the task that the tool is currently doing. If the user would like to get more information, she can click on the "Information "button. The user will then be presented by a scrollable text area that shows the current action (e.g. Detecting device ... dev 40). This text area is hidden by default so that it does not get into the way of the user while using the tool. In case of a crash occured, the text area will show more information to the user about what are the possible causes of the crash and the progress of the tool so far.

Another part of our second user study was to show the participants of our study the mockups we designed for the tool itself so that we figure out which design would be the best fit. Figure 3 shows the various mockups that showed to the participants of this study. The feedback we got from the users allowed us to make the following conclusions:

1. Almost all users liked having helpful descriptions for the various kernel features they can select.
2. Easy navigation through the various categories of features was also on top of thier desirable design features.
3. The fewer the number of categories were, the easier it was for the users to find features.
4. Some users did not like the wizard-based design because it did not offer easy navigation through features and that search was not provided in that design.
5. Most users liked the fact that they will be presented by a summary of the configuration file at the end of the process, with the ability to review it and maybe modify it before quitting the application.
6. Although many users were not in favor of the wizard-based design, they did like the question format that was presented in that design. The reason is that it gave them a better understanding of the effect of selecing/deselecting features.

Based on our conclusions, we designed a prototype for the *lkc tool* as shown in Figure 4. The figure shows various steps during the process of a configuring a Linux Kernel. The design of the prototype is a mixture between the tree-based design (Figure 3c) and the wizard-based design (Figure 3d). The rational behind this mixture is that we tried to get the best out of the two designs because they were the two most appreciated designs among the participants of our second user study. Having such design allows users to go through the step-by-step configuration process that the wizard design. In the meanwhile, users can still enjoy the comfort of free navigation through the various categories of features using the left panel. We also tried to provide the users with as few categories as possible and label them with less cryptic names than the ones used in current Linux kernel configuration tools (e.g. xconfig). No matter how the user navigates through the categories of features, she will always be presented with a question related to the selected currently viewed category. The answer to this question reflects on enabling the related kernel features accordingly. Therefore, the user does not have to directly deal with setting/unsetting kernel features, but rather answering questions that better match her usage of the machine that she wants to compile this kernel for. The menu bar at the top of the application window provides the user with fast access to tasks like:

1. Starting a new configuration process.
2. Opening a previously saved configuration file into the tool.
3. Saving the current configuration to a file on disk.
4. Keyword searching within the feature names, descriptions and help text.
5. Switching to an advanced mode, where the user will be able to use *xconfig* instead of our to complete the configuration.
6. Navigating through the categories of features presented in the left panel.

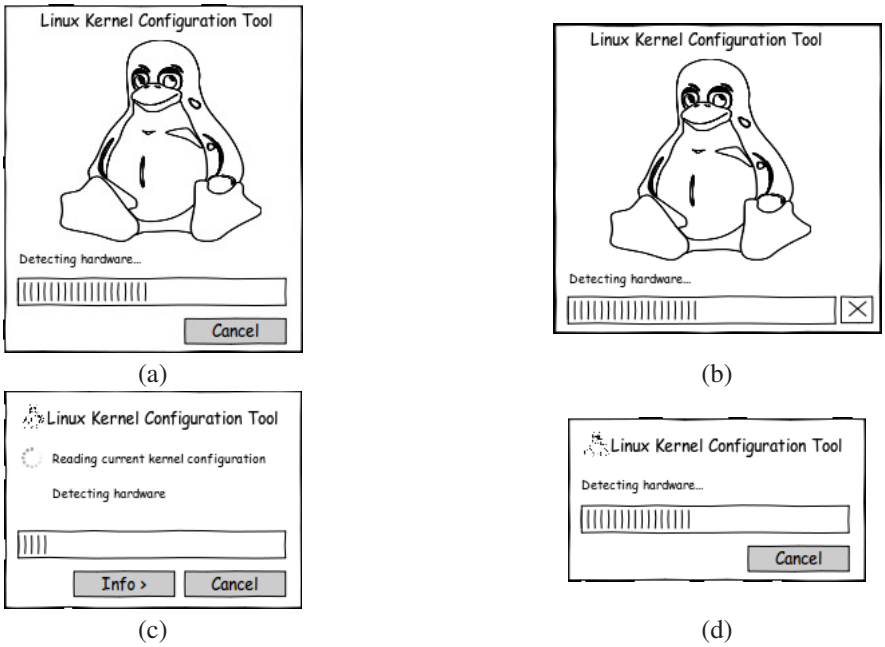

Figure 1. Splash screen design mockups

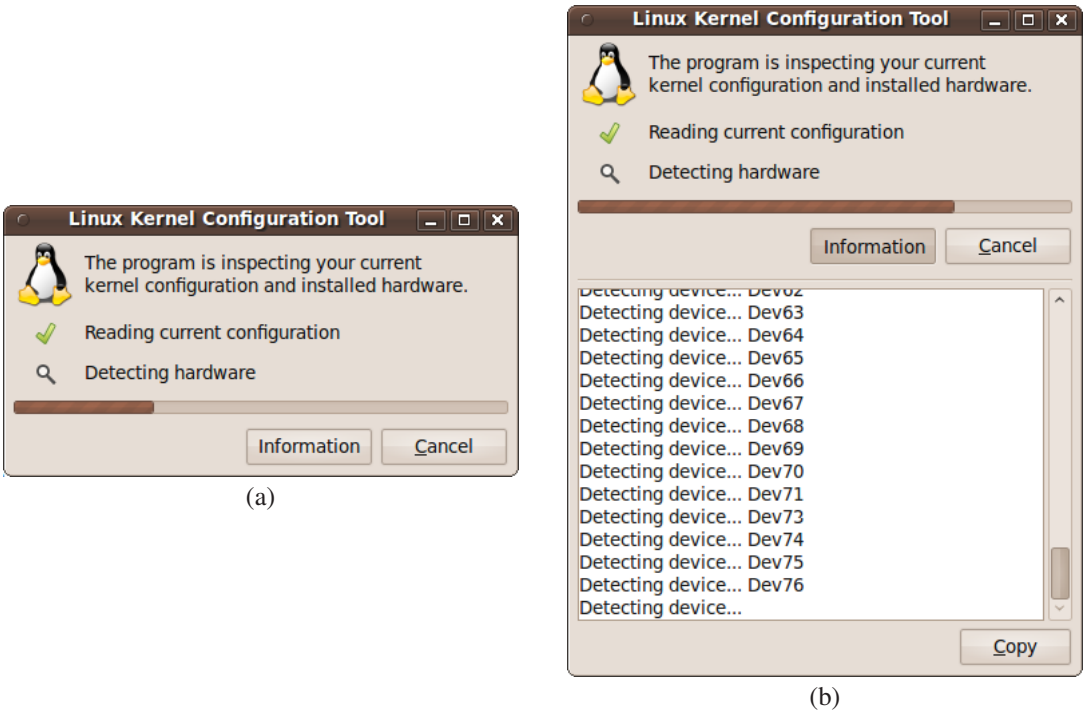

Figure 2. Final splash screen prototype

We also added to the prototype a panel that shows the statistics of the kernel that will be compiled using the currently selected features. Statistics includ:

1. Progress of the current configuration session.

2. Total kernel size based on the currently selected set of features.

3. Total number of features.

4. The stability of the kernel to be compiled by the generated configuration file. This is based on the least stable selected kernel feature.

In addition to statistics about the whole kernel, the prototype also shows information about each feature when selected.

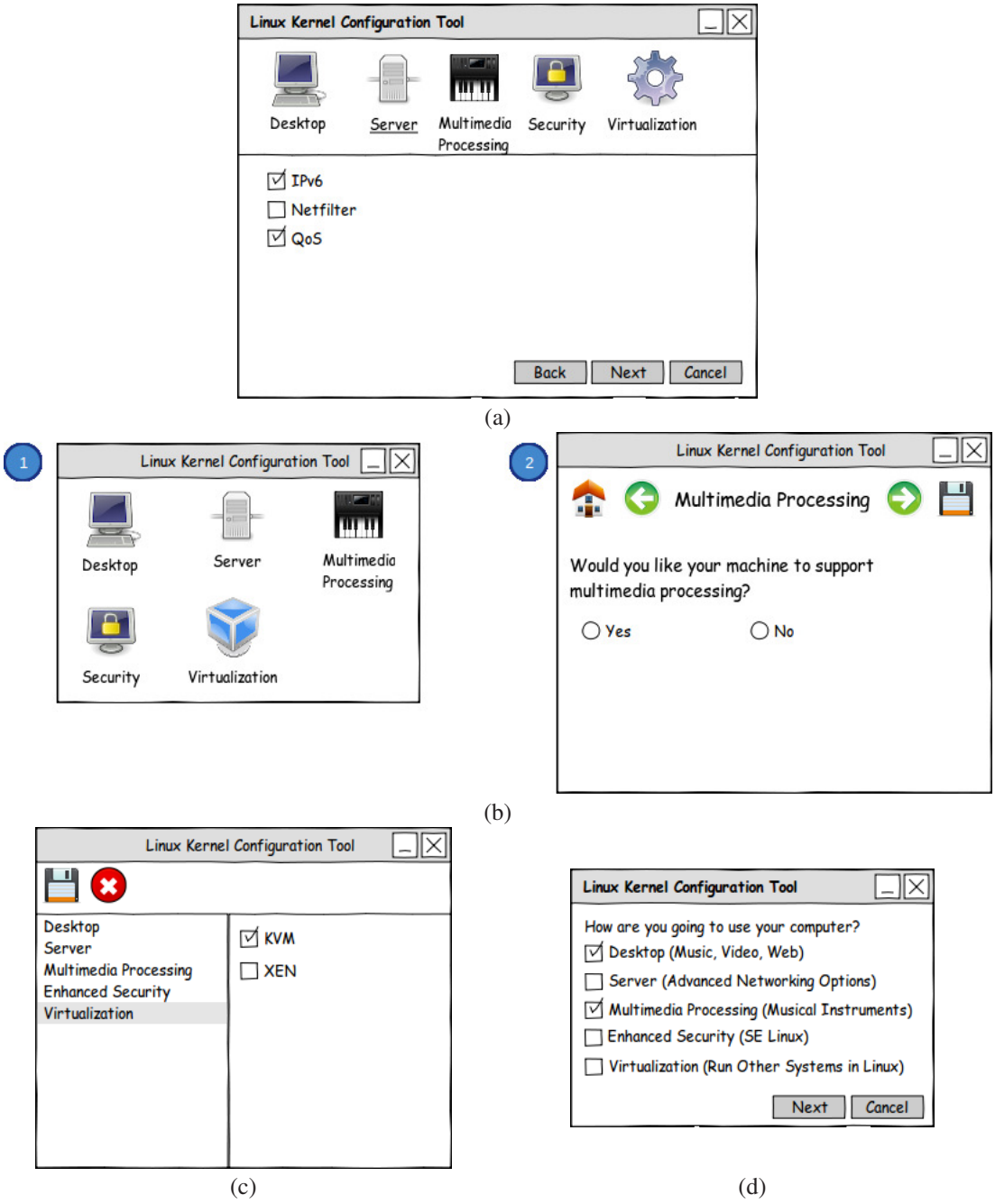

Figure 3. ***lkc tool*** **design mockups**

The information is about the effect of selecting this feature on the total kernel size, and how stable that feature is. Moreover, the prototype shows a panel at the very bottom that lists all the conflicts that are currently found in the configuration. However, this feature has not been implemented yet due to time constraints.

The first step that the user will encounter using our tool is the welcome screen (Figure 4a) where there is an explanation of how the tool can be used to configure a Linux kernel. The explanation includes some notes about the defaults options, how to navigate through categories, and what to do when the user is done. Depending on the selections that the user make in the subsequent screens, some categories of features (from the left navigation panel) will be skipped because they are marked as irrelevant to the user choices. However, if the user thinks otherwise and would like to change some of the features in such categories, she can navigate to any of those categories using the left navigation panel. Once the user feels comfortable with the configuration, she can save the configuration to a file on disk using the Save button in the menu bar at the top. She can then exit the application if

she wishes.

The implementation of our *lkc tool* prototype seperates the graphical user design from the code that actually manages that design. We used Glade [8], a rapid application development tool for GTK+ and GNOME, to design the interface of the prototype. Glade is Free Software released under the GNU GPL License, which perfectly goes along with the topics we discussed in our class. Glade has many language bindnigs making it easy to re-use the same Glade design with different languages. We chose to work with the Java bindings for Glade [9] because we are more familiar with that language which will make the process of protoyping easier and faster. We used Eclipse [6] as our integrated development environmemnt (IDE) to produce the Java code that will manipulate the interface designed with Glade.

All the work that we have done so far for this project including: source code files, mockup designs, reports, presentation and this report are avialable online at our github repository [10].

EVALUATION

Threats to Validity

External Validity

Internal Validity

FUTURE WORK

RELATED WORK

All the Linux kernel configuration tools are front-ends built on top of a single engine called Linux Kernel Configurator (LKC). LKC analyzes kernel variability and supports users during the configuration stage. LKC uses internally the KConfig language to represent variability and dependencies among features. As several researchers [17, 16] showed, there is a direct correspondence between well-understood feature models and practically crafted variability model used by the Linux kernel. In contrast with formal feature models, LKC is not supported by any formal reasoning engine (e.g. SAT-solver). This is a serious problem, because users can unconsciously create wrong configurations while having no clue why a particular configuration does not work. The Linux kernel community can benefit from adopting well-researched feature models, and make their tools more reliable and usable.

Improvements of kernel configuration tools were already introduced in 2002 when Eric S. Raymond presented the CML2 configuration system [15]. It allowed for effective reasoning on kernel feature model and also provided progressive-disclosure. There was a long debate and flame war about using that system [12]. Finally, it was rejected for various reasons, such as Eric S. Raymond's attitude, Python dependency, complexity, radically new design. Many developers preferred to introduce gradual updates instead of applying one big patch.

eCos [3] is an open source real-time operating system. It was designed to be configurable and to run on a wide variety of hardware platforms. Similarly to the Linux kernel, eCos comes with configuration tools. The eCos configurator, is much more advanced, and arguably better-thought, since it can detect conflicting options and guide users to resolve them. The eCos Configuration Tool has many features, but presents them in a single tree hierarchy. In comparison with our tool, eCos features are often low-level and very technical. The eCos Configuration Tool also does not offer any wizard, so users must have expertise if they want to skip irrelevant categories.

Hardware autodetection and kernel configuration is a recurring problem [5, 18]. Many users complain about the lack of it. The situation is slowly improving as developers added the localmodconfig [2] target to Linux-2.6.32. The command detects current kernel configuration and applies the same options to the new kernel. It is a step ahead, but the tool is very simplistic and assumes that all the required modules are already loaded. For example, if a computer has built-in Bluetooth, but the module is not loaded, the new kernel will not support the Bluetooth device. Furthermore, the autoconfiguration script offers a very coarse level of options granularity, e.g. if a computer has one sound card, such as Intel HD Audio, the script will select all available sound card drivers and all Intel HD Audio modules. The autoconfiguration tool reads configuration of the running kernel instead of detecting the actual hardware, e.g. for a laptop with the Core Duo 2 processor, it selected 686 processor in the new kernel, because that processor is selected in the running kernel. Technically, many problems with hardware detection have been solved and modern Linux distributions load drivers automatically. This is a great feature of dynamic configuration, so we decided to include hardware detection into our tool and apply to static kernel configuration.

Debian GNU/Linux device driver check & report [11] is an interesting project that helps with matching kernel modules with hardware. It reads output of lspci command, and returns a list of driver names. The project is in fact a big database that stores user's knowledge about the hardware. It could be utilized by the configurator to automatically select relevant modules if autodetection fails. Currently, the database aggregates only information about PCI devices, and does not support ISA, USB, IEEE1394 or other hardware.

System configuration is a necessary, but uneasy process. Our tool tried to simplify it by asking as few questions as possible and targeting specific group of users. The same direction is suggested in [7]. The author goes even further saying that software shall configure itself automatically. We agree with many points from the post, but making a configuration-free kernel on any platform seems rather impossible due to extra knowledge that is not included in configurators.

Free and Open Source Software usability has recently got attention from both academia and FOSS community. Several researchers [4, 14] tried to understand why FOSS developers traditionally gave low priority to usability issues. Other works studied particular projects, such as Apache [1] or Mozilla [2] to learn about social interactions, FOSS culture and the way the software is developed. Cooperation

[2]More info at: `http://bit.ly/cPgq8R`

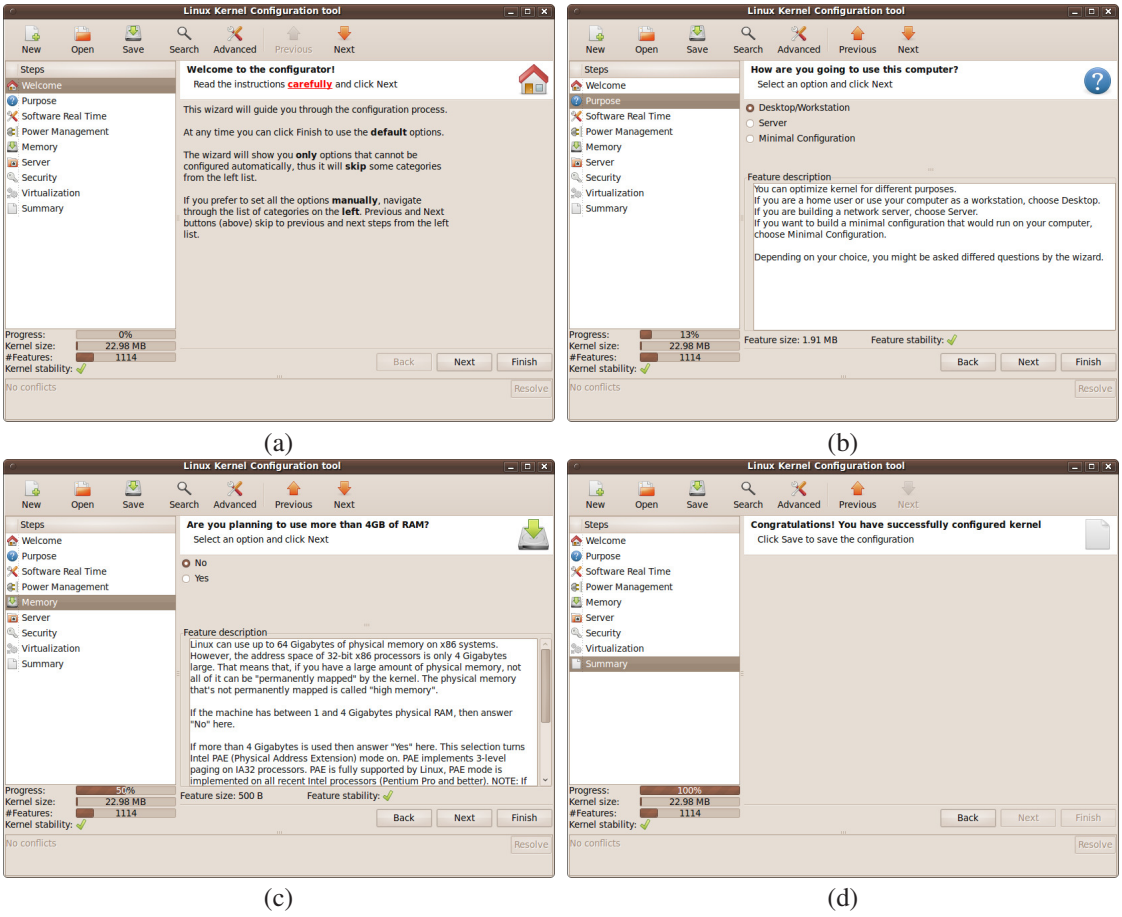

Figure 4. Final *lkc tool* prototype

of the two communities allows usability people to understand attitudes of FOSS developers. FOSS community, on the other hand, benefits from learning usability knowledge and evaluation techniques.

CONCLUSION

REFERENCES

1. J. H. Audris Mockus, Roy T. Fielding. A case study of open source software development: the apache server. In *ICSE'00*, pages 263–272, 2000.
2. J. H. Audris Mockus, Roy T. Fielding. Two case studies of open source software development: Apache and mozilla. *ACM TSEM*, 11(3):209–346, 2002.
3. J. D. Bart Veer. *The eCos Component Writer's Guide*. 2000.
4. M. B. T. David M. Nichols. The usability of open source software. *First Monday*, 8(1), 2003.
5. Debian User Forums. Is there an automatic .config generator for kernel 2.6.33.4?, 2010. *Accessed: July 28, 2010.*
6. Eclipse Foundation. Eclipse, 2010. *Accessed: July 28, 2010.*
7. Frank Spillers. Configuration Hell- The Case for the Plug and Play User Experience, 2010. *Accessed: July 28, 2010.*
8. GNOME. Glade - A User Interface Designer, 2010. *Accessed: July 28, 2010.*
9. GNOME. The java-gnome User Interface Library, 2010. *Accessed: July 28, 2010.*
10. Kacper Bak, Karim Ali. lkc, 2010. `http://github.com/kbak/lkc`.
11. Kenshi Muto. Debian GNU/Linux device driver check & report, 2010. *Accessed: July 28, 2010.*
12. Kernel Trap. Linux: CML2, ESR & The LKML, 2002. *Accessed: July 28, 2010.*
13. Linux-Kernel. Aunt Tillie builds a kernel (was Re: ISA hardware discovery – the elegant solution), 2002. *Accessed: July 28, 2010.*
14. Morten Sieker Andreasen, Henrik Villemann Nielsen, Simon Ormholt Schroder. Usability in open source software development: Opinions and practice. *Information Technology and Control*, 35(3):303–312, 2006.

15. E. S. Raymond. *The CML2 Language: Constraint-based configuration for the Linux kernel and elsewhere.* 2000.

16. S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. Variability model of the linux kernel. In *VaMoS'10*, pages 45–51, 2010.

17. J. Sincero and W. Schröder-Preikschat. The linux kernel configurator as a feature modeling tool. In *ASPL'08*, pages 257–250, 2008.

18. Soft32 Linux User Forums. kernel autoconfig option?, 2007. *Accessed: July 28, 2010*.