Hardware Detection for GNU/Linux

Josh Triplett

June 2, 2004

Contents

Copyright © 2004 Josh Triplett.
This document is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 2 dated June, 1991.

Abstract

This document discusses the design of a comprehensive hardware detection and configuration system for the GNU/Linux operating system.

GNU/Linux is a Free Software operating system, popular among technical users because it grants the freedom to study, modify, and distribute the system. Over time, GNU/Linux is rapidly improving in usability for non-technical users as well. One major step in that direction would be the introduction of a system to automatically set up new hardware for use.

When computer users plug in a hardware device, they expect it to start working immediately, with as little manual configuration as possible. Users shouldn't need to understand how their computer interacts with the device; they only want to run an application that allows them to use their hardware. Achieving this level of functionality requires a hardware detection and configuration system. This system will:

- Detect new, unconfigured devices
- Gather all necessary information to configure them, asking the user for only the information that cannot be automatically determined
- Perform any necessary setup
- Run an appropriate application for the user to interact with the device

This document discusses the design of such a system, including architectural factors that affect the design, currently-available components that will form parts of the system, and the work required to create a finished solution.

Introduction

GNU/Linux is a Free Software operating system, meaning that everyone has the right to use it, copy it, modify it, and distribute it. (Free Software Foundation, 2004) This freedom allows communities of developers from all parts of the world to work on each part of the system collaboratively. Users of the system choose it due to both the freedom it offers and the high-quality software this freedom promotes.

For the first few years after GNU/Linux was first written, it was primarily usable only by developers and hobbyists - those who were more than willing to learn much about how the system worked in order to use it. As GNU/Linux matured, however, developers began to seriously work toward usability for the average computer user, who doesn't really care how the system works as long as it does. This has led to such advances as Human Interface Guidelines to define how to interact with the user (Benson et al., 2002), and usability studies to determine the effectiveness of that interaction (Relevantive AG, 2004). However, one notable usability feature that GNU/Linux currently lacks is a comprehensive hardware detection and configuration system.

When users plug in a new device, they expect it to start working as soon as possible, with minimal user input required to set it up. This functionality requires hardware detection and configuration software that can detect the new device, ask the user for any required information that cannot be automatically determined, perform any necessary setup, and run an appropriate application to use the device. This document discusses the design of such software, including architectural factors that affect the design, currently-available components that solve parts of the problem, and work still needed to create a finished solution.

Architecture of GNU/Linux

GNU/Linux systems have some fundamental architecture choices that affect the design of any potential hardware detection system.

First, a GNU/Linux system supports multiple independent users, each with their own private directory for files. These users can all access the system simultaneously, either through a network connection, or by using multiple monitors, keyboards, and mice on a single system. A user typically has permission to write only to their own directory and to temporary space, and cannot modify system files or other users' files. There is one administrative user, called "root", who has permission to write anywhere, and to directly access system hardware. Even on a single-user system, the administrator and sole user typically uses a normal user account for day-to-day computing, and only uses the root account for administration. Therefore, the hardware detection system must ensure the user has sufficient permissions to access a given device and configure the necessary software support, and enable that user to determine which other users will be allowed to use the device.

Second, a GNU/Linux system can have many different graphical environments installed, or none at all. The most popular environments are KDE (KDE e. V., 2004) and GNOME (GNOME Project, 2003), each based on a different underlying graphical framework. Typical GNU/Linux systems have thousands of graphical applications to choose from, most based on one of these graphical frameworks and designed to fit well in the corresponding environment. The hardware detection system must integrate into the user's environment of choice, while having as little environment-specific code as possible.

Finally, GNU/Linux is highly modular and layered. The three layers relevant to a hardware detection system are drivers, daemons, and applications. Drivers are the small portions of code that run in the kernel and provide a low-level interface to hardware devices. Daemons are programs that access that interface and allow applications to perform useful operations with the devices while being less dependent on the actual device type. (Daemons are not necessary for all devices.) Applications are the myriad user programs that can take advantage of hardware devices, by accessing the interface provided by drivers or daemons. The hardware detection system must interact with all three layers in order to fully configure a piece of hardware.

Current State of the Art

Current hardware detection systems have excellent support for detecting the appropriate low-level kernel driver to use. These systems have databases that map identifying information about the hardware to names of drivers. When a new device is plugged into the system or detected at boot-time, the system looks it up, finds the driver name, and loads that driver.

At the daemon level, hardware detection is less mature. Many daemons provide easy-to-use setup utilities, but the user must still provide much configuration information that could be determined automatically. For example, the Common UNIX Printing System (CUPS) and some of its frontends have guided configuration wizards to help the user detect and set up a new printer easily, but the user must still select the type of printer they have, name it, and possibly download a PostScript Printer Definition (PPD) file. For some printers, the user may also need to install and additional software.

At the application level, current applications can generally determine an appropriate default device to use, such as the system default printer, scanner, or sound card. If the user wants to use a device other than the default, they can generally choose the device they want from a menu of all devices on the system. However, the user must still manually run the appropriate application, and so they must know what application to use for each type of device, forcing them to think in terms of programs to run rather than tasks to perform. In addition, if the user has never used a particular type of device before, they may not have the appropriate software installed to use it, so they must determine what applications to install and install them. Some GNU/Linux distributions choose to avoid this by installing everything the user might need initially, but this makes for a lengthy initial installation, and leads to the problem of finding the appropriate application in a huge menu of choices, many of which may never be used.

Ideal Scenario

When a user plugs in a new hardware device, the hardware detection system will attempt to set up the device while asking the user for as little information as possible. For many devices, all required configuration information can be detected automatically without asking the user, so the system will simply set up and enable the device and notify the user unobtrusively when the device is ready for use. Depending on the device, the configuration utility may need to request administrator credentials. The user may choose to receive a prompt first, showing the system's planned setup and asking if the device should be set up automatically, allowing the user to change some options or configure the device manually. The user and the GNU/Linux distributor will choose whether this prompt appears by default or whether devices are configured with no interaction.

For devices requiring some user input to set up, the hardware detection system will notify the user that it found a new device, and allow the user to open a step-by-step configuration window. This configuration window will request administrator credentials if necessary, and then ask for the information required to set up the device. The user should always have the option of more control over the configuration, but they must have the option of simply accepting a set of defaults. The hardware detection system will then configure all necessary drivers and daemons needed to access the hardware, and help the user install applications to use the device if they are not already installed.

Once the device is set up, or if a device has already been seen and configured before, the hardware detection system will offer to launch an application to work with the data. This could be a scanning program to operate a scanner, a file manager to view the contents of a disk, or a save/print pictures wizard for a digital camera.

Modularity, Layering, and Existing Software

Like GNU/Linux itself, the hardware detection system will be modular and layered, both to handle many different frontends for different environments, and to take advantage of existing software that can provide some of the components. These components and their interactions are shown in Figure 1.

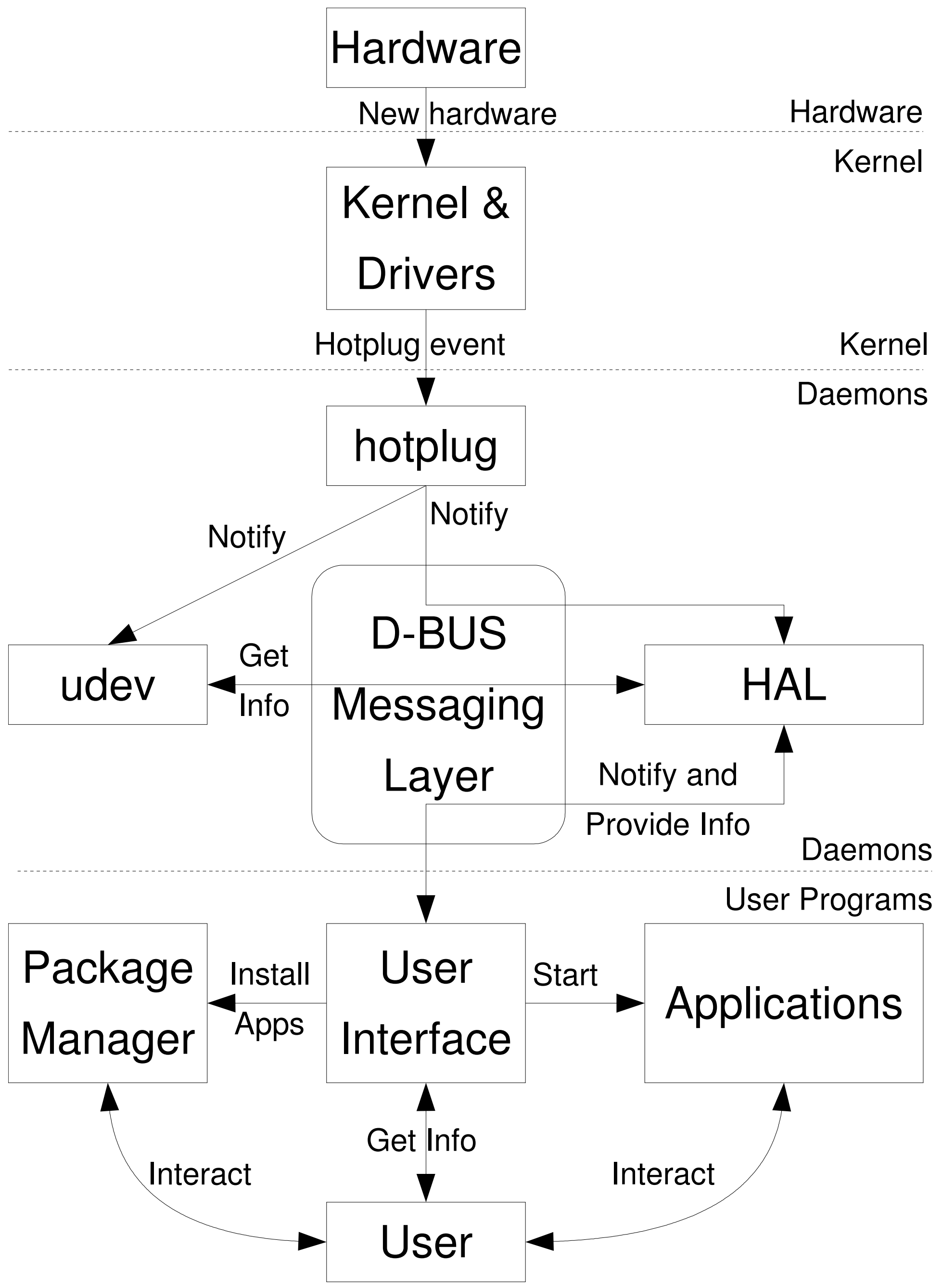

Figure 1: Components and Interactions

Hotplugging

In order to set up a new piece of hardware, the hardware detection system needs some way to detect when a new device is plugged in and take appropriate action. During the creation of the Linux 2.4 kernel, developers added support for hotplugging hardware such as USB devices and PCMCIA cards, meaning that they can be inserted while the system is running. When the kernel detects a new device, it calls a program called hotplug, which can then do some minimal level of configuration, such as bringing up the network when a network card is inserted (if the network supports automatic configuration). (Linux Hotplug Project)

This works well for devices designed to be hotplugged, but what about devices that are present when the system boots? To handle this scenario, the newest version of the kernel, Linux 2.6, can call hotplug for any device, even if it was present at boot time; this is known as "coldplugging". This unifies the low-level architecture of the hardware detection system, so that it does not need two separate backends to handle normal and hotplugged devices.

Once the hotplug utility gets control, it can run any arbitrary command scripts set up previously. Current scripts simply enable a device if the user has previously set it up. However, the same mechanism also allows the next layer of the hardware detection system to hook in and get a notification for each new device.

Device Identification and Naming

Once a new device has been detected, the system must identify and name it. In a GNU/Linux system, every device appears as a file in a special directory, `/dev`, and applications access devices through the corresponding device file. On many systems, that directory contains thousands of device files, most for devices the user does not have. This architecture was designed when hardware rarely changed, and so rather than creating new device files for new devices, it was easier to have every possible device file available. This system also names devices based on how the system accesses them: `/dev/sdc` is always the third SCSI disk, and `/dev/lp0` the first printer.

Associating a particular device file with the third hard disk or the first printer works fine for static devices, but what happens when the user plugs in devices in a different order? The user should not need to figure out if the USB camera they just plugged in is `/dev/sda` or `/dev/sdb`. Ideally, a particular name will be associated with each unique device, such as `/dev/minolta-camera`.

The solution to this problem can be found in udev, another part of the hotplug suite. udev provides an easy way to configure device naming based on the identity of a device. This naming configuration can be provided by simple configuration files, or by running other programs that determine the appropriate name. This allows the hardware detection system to hook into the device naming process. (Linux Hotplug Project)

Hardware Abstraction

The low-level layers of hotplug and udev will interact directly with the kernel interfaces to the device. However, these layers should not maintain the database of hardware information, because they are specific to the GNU/Linux operating system, and the hardware detection system as a whole must be portable. In addition, more information may be needed than is immediately available to the low-level layers, and it may be necessary to look at multiple sources to collect all the information needed to set up and use a device. Finally, user-level applications should not need to know how a device is connected in order to access it.

The Hardware Abstraction Layer (HAL) solves these problems in a particularly elegant way. HAL provides a view of all devices on the system, with a set of properties for each. These properties can contain any arbitrary information associated with the device, from any source that can communicate with HAL. HAL uses an XML-based database of Free Device Information (FDI) files to map the basic identifying information from a device to information about the type and capabilities of the device. As described by the HAL Frequently Asked Questions (FAQ) page, HAL can merge this information gathered from the hardware with information from the kernel, system configuration files, data from user interaction, and many other sources. Other layers of the hardware detection system can query HAL to determine the information needed to set up the device, while daemons, applications, and hardware access libraries can query HAL to decide what devices to act on. (HAL)

User Interaction

The device configuration process may require user interaction, and so will launching the appropriate application once the device is configured. The low-level layers cannot simply open a window to interact with the user, because there may be several users on the system, each running a different desktop environment. The user interface must appear only for the appropriate users, and must integrate into those users' environments. In addition, the user-interface should run as a normal user for security reasons, while the low-level layers will run as root to access the hardware. For these reasons, all interaction will occur through a user-interface layer, which must somehow communicate with the rest of the hardware detection system.

This is where D-BUS comes in. D-BUS (for Desktop Bus) is a messaging layer, which allows applications to send messages to each other, and listen for certain types of messages. (D-BUS) The hardware abstraction layer can use D-BUS to send a notification about new devices, while the high-level layers can listen for such notifications. This would allow a different user-interface layer for a different desktop environment to communicate with the same low-level layers. Ideally, as little of the overall system as possible should specific to a particular implementation of the user-interface layer.

D-BUS also solves another problem: what to do if a device is plugged in when no user is logged in, or if devices are detected on boot before users can log in. When a user logs in and the user-interface layer starts up, it can ask the other layers (via D-BUS messages) if there is any new hardware to configure, and then configure it as if it was just plugged in.

Reviewing and Configuring Current Hardware

In addition to setting up newly-plugged-in hardware, the hardware detection system must allow the user to review all devices and change their configuration. This includes the hardware that is currently plugged in, as well as the saved hardware configurations for devices seen in the past. The hardware management utility should allow the user to see the questions they answered when the hardware was initially plugged in, and change the answers, with the device configuration updated accordingly. While the hardware reconfiguration interface should be part of the user-interface layer in order to integrate with the user's environment, the underlying reconfiguration mechanism should be uniform for all frontends.

In the past, many efforts to create user-friendly configuration utilities have met with backlash from advanced users, because such utilities frequently assume they are the only application performing system configuration. Based on this assumption, these utilities overwrite existing configuration files when run, and do not preserve configuration changes made outside of the utility. They also output long, unorganized configuration files with no comments, making further editing difficult. GNU/Linux has a long history of being configurable using only a text editor, and any configuration utility must preserve that functionality for users who wish to use it. For this reason, the reconfiguration utility must preserve user changes to configuration files, in order to support both novices and experts.

Portability Concerns

In addition to the most common personal computer platforms, x86 and PowerPC, the GNU/Linux operating system runs on architectures ranging from PDAs to mainframes. The most widely ported GNU/Linux distribution, Debian, currently runs on 11 architectures, with several more on the way. (Debian Project, 2004) The hardware detection system should endeavor to support the users of these architectures.

For architectures that support the same hardware standards as a PC, such as PCI, USB, and Firewire, portability simply requires the ability to compile and run on such machines, and detect the same types of hardware, using the same kernel interfaces. Other, more unusual architectures may require additional support for new types of hardware specific to that architecture. In all cases, only the low-level layers that directly access the hardware will change; the hardware abstraction layer will insulate the high-level layers from these changes.

In addition to different hardware architectures, the hardware detection system should be portable to other Free Software operating systems, including FreeBSD (FreeBSD Project, 2004), NetBSD (NetBSD Foundation, Inc., 2004), OpenBSD (OpenBSD, 2004), and even GNU/Hurd once it is ready for non-developer use (Free Software Foundation, 2003). Supporting a different operating system will require many changes in the low-level layers of the hardware detection system, to accommodate the differences in the kernel and driver, but the high-level layers should still work unchanged.

Work Needed

There are several areas in which major work is still required to make the hardware detection system complete. These include gathering device information, adding low-level support for more types of devices, writing user-interface layers, and installing applications on demand.

Gathering Device Information

In order to be useful, the hardware detection system must know about as many devices as possible. Much of this information is already available in various hardware databases. For example, LinuxPrinting.org currently maintains a database of nearly 1,200 printer models and the corresponding driver names, along with PostScript Printer Definition files for each printer. This database is readable both by users and by programs. (Taylor and Kamppeter, 2004) The hardware detection system could add a module to read device information from such sources, or the information could be automatically converted into the native format of the hardware detection system.

In the standard Free Software development model, preliminary versions of the software are made available for use by developers or advanced users. When this occurs, many of these early adopters will submit device descriptions for the hardware they personally have access to. In this way, many hardware devices, from the popular to the obscure, will be well-supported by the time the software is considered stable enough for use by the average user.

Once the device information is available, it could also be used to generate a comprehensive hardware compatibility list. Such a list would allow users to evaluate the support for their current hardware when switching to GNU/Linux, or determine what model of device to purchase for their GNU/Linux system.

Low-level Device Support

While adding support for a new device is simply a matter of providing some information about the device, adding support for a new type of device requires some work on the low-level layers. Each new device type, connection type, or daemon requires some low-level support in addition to the new information about each device. Because of the hardware abstraction layer, these changes should not affect other components of the hardware detection system.

User-Interface Layers

Each desktop environment needs a user-interface layer to interact with the user and define the device actions specific to that environment. Project Utopia (Love, 2004) is currently working on an interface for the GNOME desktop environment. A similar effort is needed for the KDE environment, as well as a possible generic layer to handle the many users who prefer a less common environment. In theory, a user-interface layer could even be provided for non-graphical users, who interact with the system only through a text console. Such an environment could help novice users setting up a headless server for their home network, or system administrators who are not necessarily hardware experts.

Installing Applications on Demand

While the initial versions of the hardware detection system can simply rely on all necessary software packages being installed, eventually the system should have support for installing applications as they are needed. When the user adds a new hardware device, the hardware detection system can look up useful applications based on the type of the hardware device. This process will also involve some environment-specific policy, since applications designed for a specific environment will integrate better into that environment.

The exact procedures for querying available applications and installing applications differ between GNU/Linux distributions, but the basic idea is generally the same. There is an underlying package-management architecture for each distribution, which provides tools for the user to browse available packages, install and remove packages, or upgrade to the latest versions. The hardware detection system can have a distribution-specific layer for managing packages, which can query the available packages, provide them as suggestions to the user-interface layer, and start up the user's preferred package manager to install them if desired. This functionality will also require support from the individual distributions, by labelling packages that would be useful for using a particular type of device in a particular desktop environment.

Implementation of Ideal Scenario

When a user plugs in a new hardware device, the kernel will detect the device and generate a hotplug event. The hotplug program will search for the type of device, and run an appropriate script. This script will notify HAL of the new device via D-BUS. Hotplug will also notify udev, which will ask HAL for information needed to name the device; this information may not be available yet, so udev will have to wait. HAL will gather as much information as it can, both directly and via external programs. Once all possible information has been gathered and various HAL device properties have been set, the user-interface layer is notified about the device. The user-interface layer decides, based on a policy defined by the desktop environment, distributor, and user, what to do about the device. If there is enough information to set the device up non-interactively, the user-interface layer may do so; otherwise, it may guide the user through providing the remaining information needed. At this point, there is enough information to set up the device node via udev. The user-interface layer may also determine if additional applications are needed, and offer the user the option of installing them. Finally, once the device is set up, the user-interface layer will determine the appropriate action to take, such as launching an application to use the device.

Conclusion

GNU/Linux has come a long way, maturing into a system useful to users with a broad range of experience. However, hardware setup under GNU/Linux is still primarily a manual process. This makes such setup difficult for new users, and time-consuming even for experienced users. The addition of a comprehensive hardware detection and configuration system would greatly improve usability for novice and expert alike.

Glossary

application Can refer to any program, but commonly refers to a program that the user interacts with directly. To access hardware, applications talk to daemons or device files.

coldplug To synthesize a hotplug event for a device present when the computer boots, in order to use the same hotplug architecture for every device.

daemon A program that runs outside the kernel, talks to devices through device files, and provides a high-level interface for applications.

D-BUS Desktop Bus - a messaging layer, allowing programs to send and receive messages without talking to each other directly.

device A piece of hardware.

device file A file, usually in the `/dev` directory, which represents a device. Programs can access the device by reading and writing this file.

driver A low-level hardware access program, which runs in the kernel, talks directly to a hardware device, and provides an interface for applications to access that device, typically through a device file.

Free Software Software that grants users the right to use, copy, modify, and distribute it.

GNOME The GNU Network Object Model Environment - one of the two major desktop environments for users of GNU/Linux.

GNU GNU's Not UNIX - a project to create a complete, UNIX-compatible operating system composed entirely of Free Software.

GNU/Linux A Free Software operating system based on GNU software running on the Linux kernel.

HAL Hardware Abstraction Layer - a layer between devices and other programs, which provides a uniform interface for gathering information about devices.

hotplug To plug in a device while the system is running. Also refers to the program of the same name, which detects hotplug events in GNU/Linux.

KDE K Desktop Environment - one of the two major desktop environments for users of GNU/Linux.

kernel The core of an operating system, which runs directly on the hardware and provides an interface to run higher-level programs.

Linux A Free, UNIX-compatible operating system kernel.

operating system The set of all programs that run all the basic operations of a computer, and provides services to applications.

udev A GNU/Linux program that manages the creation and naming of device files.

UNIX A class of operating systems characterized by the interface they provide to applications, as well as certain common architectural choices. GNU/Linux, FreeBSD, NetBSD, OpenBSD, and GNU/Hurd are all UNIX-compatible operating systems.

Works Cited

Calum Benson, Adam Elman, Seth Nickell, and Colin Z Robertson. GNOME Human Interface Guidelines, 2002, 24 May 2004. ⟨http://developer.gnome.org/projects/gup/hig/1.0/⟩.

D-BUS. D-BUS, 24 May 2004. ⟨http://dbus.freedesktop.org/⟩.

Debian Project. Debian GNU/Linux - Ports, 2004, 24 May 2004. ⟨http://www.debian.org/ports/⟩.

Free Software Foundation. The GNU Hurd, 2003, 24 May 2004. ⟨http://www.gnu.org/software/hurd/hurd.html⟩.

Free Software Foundation. The Free Software Definition, 2004, 24 May 2004. ⟨http://www.gnu.org/philosophy/free-sw.html⟩.

FreeBSD Project. The FreeBSD Project, 2004, 24 May 2004. ⟨http://www.freebsd.org/⟩.

GNOME Project. GNOME: The Free Software Desktop Project, 2003, 24 May 2004. ⟨http://www.gnome.org/⟩.

HAL. HAL - Hardware Abstraction Layer, 24 May 2004. ⟨http://hal.freedesktop.org/⟩.

KDE e. V. KDE Homepage - Conquer your Desktop!, 2004, 24 May 2004. ⟨http://www.kde.org/⟩.

Linux Hotplug Project. Linux Hotplugging, 24 May 2004. ⟨http://linux-hotplug.sourceforge.net/⟩.

Robert Love. Robert Love's Log: Project Utopia, Apr 2004, 24 May 2004. ⟨`http://primates.ximian.com/~rml/blog/archives/000395.html`⟩.

NetBSD Foundation, Inc. The NetBSD Project, 2004, 24 May 2004. ⟨`http://www.netbsd.org/`⟩.

OpenBSD. OpenBSD, 2004, 24 May 2004. ⟨`http://www.openbsd.org/`⟩.

Relevantive AG. Linux-Project: Studies, 2004, 24 May 2004. ⟨`http://www.relevantive.de/Linux-Usabilitystudy_e.html`⟩.

Grant Taylor and Till Kamppeter. LinuxPrinting.org, 2004, 24 May 2004. ⟨`http://linuxprinting.org/`⟩.