A Case Study in Open Source Patch Submission

Studienarbeit in Computer Science

by

Holger Macht

born on 1982-08-18 in Hof a.d. Saale

at

Department of Computer Science
Professorship for Open Source Software
Friedrich-Alexander University Erlangen-Nuremberg

Advisor: Prof. Dr. Dirk Riehle

Start: 2011-11-01
End: 2012-05-25

I affirm that this thesis is my own work and has never been used before, for any auditing purposes. All used sources, additional used information and citations are quoted as such.

Nuremberg, 25th May, 2012

Holger Macht

Abstract

Although open source and the related processes have become an inherent part of the computer industry, companies and other contributors are often reluctant when it comes to active involvement and participation. The reasons are manifold. Strong ones are that most open source projects are self-governing, without a fixed road map or schedule, each which its own development process. This seems to make them unreliable and unpredictable, because personal intentions cannot be enforced and rely on the open source community in charge of leading the projects.

Within an exemplary case study, a new feature spawning multiple layers of a GNU/Linux based operating system, and thus different open source projects, will be implemented and submitted for inclusion. All this bounded by a predefined, fixed time frame, which might be exemplary for schedule and budget driven company structures. The underlying development processes of the Linux kernel, a middle ware project called UDisks and the GNOME desktop will be considered and acted upon accordingly.

As a whole, the feature submission failed, because it was not possible to include all the required changes in all target projects in time. The failure has two main reasons: First, it is caused by technical problems which could occur in every software project, and thus not related to open source processes in particular. Second, and more relevant for this research, delays occurred due to common obstacles one has to face in the individual open source development processes. Individuals cannot put as much pressure as they like on the projects and are kind of at their mercy.

As an overall outcome, trying to include a new feature in different open source projects depending on each other is possible, however, unpredictable to a certain extent. The open source community has its own rules and processes, companies or other contributors cannot rely on being able to influence in whatever way they want. The advancements heavily depend on the relevant community and project members and thus the process involved. Conclusion: When submitting patches, always expect another iteration.

Contents

1. Introduction

Open source, open standards and the processes around them have become a more and more interesting alternative to proprietary software development during the last couple of years. Even in big well-known companies like Microsoft[1], which once were famous for their closed source approaches, the borders between closed and open source become blurred. The founding of a subsidiary with the goal to strengthen and incorporate more open source business [2] is only one example from the recent past. Also hardware manufacturers like Intel[3], NVIDIA[4] or Broadcom [5] become more and more engaged when it comes to creating and releasing open source drivers for their hardware components, especially for GNU/Linux based operating systems.

However, other companies are often more reluctant in this area. Open source software development often seems to endanger their well-known, proved and existing development processes, their intellectual property and last but not least, their immediate control over their own source code. This does not necessarily exclusively apply to business companies, but might also concert individual developers which aim to become contributors in the open source development community. How to deal with project members in the open source world? When, how, in which form and where to submit code? Those are common questions each single developer or company might ask.

Those answers might be relatively simple to be answered if the targeted project is rather small, same for the actual contribution. However, if bigger changes and projects are involved, maybe even different projects depending on each other, the actual situation becomes rather complicated and unclear. It can be considered a great challenge for companies or an independent open source contributor to quickly adapt themselves to those different models. Concrete and step by step guidance is often missing or sparse.

So this research project tries to fill this gap, to a certain extend and as far es possible. It tries to answer the question if it is possible, or at least predictable, to implement a new feature spawning multiple open source projects within a predefined time frame. This is intended to be accomplished with the help of a concrete, real-world example which is targeted for the open source communities around GNU/Linux operating systems. It can be considered as an additional source of information when it comes to deciding,

[1] http://www.microsoft.com

[2] http://www.h-online.com/open/news/item/Microsoft-creates-open-source-offshoot-1520424.html

[3] http://www.intel.com

[4] http://www.nvidia.com

[5] http://www.broadcom.com

whether to put efforts into open source or not. The research work does explicitly exclude other possible benefits or disadvantages of open source software development. Those discussions are not subject of this thesis. It is rather a mixture of software design and open source development methods, with a strong focus on the actual patch submission process.

After choosing an appropriate research method (cf. 3), the thesis looks at three different open source projects depending on each other. Their development process (cf. 5), including their source code availability, incorporated tools or contributor interaction plays an important role when it comes to understanding and planning the upcoming feature. In this regard, special focus lies on the project around the Linux kernel[6], because it can be seen as the foundation stone for the operating system in a broader sense. Furthermore, it is an ideal example of a very large, successful project with thousands of contributors and community members.

After gaining the needed project knowledge, a proper schedule will be aligned (cf. 6). This schedule will be the guide line for the subsequent implementation of a feature improving the dock station support within GNU/Linux based operating systems (cf. 7). Once ready, it will be submitted to the relevant projects (cf. 8) piece by piece in form of so called patch sets. Feedback requiring the resubmission of the relevant source code is expected, due to the size of the changes. The research project ends with a fixed date, or the moment the feature is officially shipped with the release of all concerning projects. The latter is not expected to come true, though. So the implementation and proper functionality of the performed feature is not entirely subject to this work. It is not enough to have this implemented and working, but to get the changes accepted in the relevant open source projects so that Linux distributions can pick them up.

On the way, all this will provide a rough guide line on how to create and submit those patches to other, but similar, open source projects. Finally, it will assess the success or failure of the outlined strategy to identify some best practices and likely successful strategies for submitting patches to large well-known open source projects.

[6] `http://www.kernel.org`

2. Literature Review

A lot has been written about Free/Libre/Open Source Software (FLOSS) and the surrounding processes the last couple of years. More and more companies get involved in open source software (OSS) development. Of course, academics are picking this topic up to conduct scientific research and release papers, books and articles. While the topic handled here focuses on the actual submission process to large well-known open source projects, most of the available material addresses the underlying principles and methodology. For instance, Gwendolyn K. Lee and Robert E. Cole describe and compare a community-based development model for knowledge creation vs. a firm-based one. The paper is titled "From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development"[9]. Similar for the research team around Audris Mockus and the paper "A Case Study of Open Source Software Development: The Apache Server"[11]. It explicitly tries to compare the OSS development process to the commercial world and concludes that hybrid forms might work best.

However, this work does not try to emphasize advantages nor disadvantages of the underlying process, it takes it as it is. As this paper primary focuses on the Linux kernel as one of the large OSS projects, the corresponding process is especially of interest. However, a lot of related documentation and information can be transformed and applied to the other projects involved in this work, too. It is not exclusively confined to the Linux kernel. For technical development documentation, the book "Linux Kernel Development"[10] by Robert Love can be mentioned at this place. But also Andi Kleen's paper "On submitting kernel patches"[8] should not remain unmentioned because it can provide good guidance for both technical patch creation and submission. An all-in-one document trying to cover both process and technical related matters is provided by the Linux Foundation[1] in form of "How to Participate in the Linux Community - A Guide To The Kernel Development Process"[5].

When focusing on the patch as single object of research, "Small Patches Get In!"[14] provides deeper insight into the relationship between emails and patches and their submission and acceptance within different open source projects.

For planning the submission process of changes targeted for inclusion into the Linux kernel, its release criteria is playing an important role. This and an evaluation of which patches are most likely accepted and which are not is provided by a paper headed with "Release criteria for the Linux kernel"[7].

[1] `http://www.linuxfoundation.org`

What most of the mentioned documents have in common is the fact that the process is described rather theoretically, as it might come true in an ideal world. However, this thesis concentrates on the actual sequence of incidents and obstacles one has to face on the way. In a way, it proves or refutes the validity of available technical process documentation for the different projects. It makes use of the already available research results, process and software documentation, tries to define a schedule for adequate large open source contributions and finally recapitulates the outcome and consecutive process. Research covering exactly this approach is rare.

3. Research Design

Choosing the appropriate research method is one of the first steps within a scientific research project. It should be chosen according to the specific research question you are asking and has imminent impact on the results and generality of the research results.

In the handled case, the chosen research topic is a real world example, driven and enclosed into the processes outlined by big open source projects. Thus, choosing an appropriate research method a priori disqualifies some of the available possibilities. One of those is the controlled experiment. According to Easterbrook et al. [6], a controlled experiment requires a clear hypothesis and an controllable environment. Usually software engineering experiments require human subjects to perform a specific task under certain circumstances, looking at certain predefined variables. As far as possible, those tasks need to be performed without heeding attention to the context, where influence needs to be minimized as much as possible. Especially because of this context, which obviously plays an important role in the considered project, this research topic cannot be performed as an experiment. Also the other mentioned attributes for an controlled experiment do not quite fit here.

Other existing research methods like survey researches or ethnographies obviously do not fit the criteria taken as a basis for this project.

Another possibility for a suitable research method would be the case study. In contrary to the controlled experiment where a clear research hypothesis is needed, the case study requires a clear research question. For the technical issue handled here, the question is if it is possible to get a new feature into large open source projects, here the Linux kernel and others, within a pre-defined time frame. This question is deeply embedded into its context and the obstacles resulting from human interaction, acceptance, code quality and correctness.

During the whole project, and for the case study's results to become valid, the project and involved human interaction needs to happen without the involved parties being aware of the fact that they, or the project they represent, are part of an ongoing research. A case study is some kind of experiment in real world and thus it needs to be taken explicitly care of to not mention this fact during any future communication. It would drastically falsify the collected data.

The used approach also fits the criteria of a typical case. It is no special concern to get a new feature into an open source project and thus improving the support for a hardware device. Companies releasing computer devices for the global market are doing this on a regular basis. A lot of hardware vendors are trying to support their hardware by software drivers written and maintained by themselves. Examples include NVIDIA

or AMD for their graphics drivers or Intel for various devices like network cards, USB or audio devices.

Another important attribute of a case study is the source of information. Qualitative data is often collected via observations. According to Easterbrook et al. [6], this needs to be performed with respect to a well-defined unity of analysis. In this case, this would be the specific development process found in the Linux kernel project. Also, the context, like people, sub-projects, time and different interests needs steady consideration. And this context is also the most important weakness of a case study. Finding the right balance between external factors and the actually performed tasks is a major challenge of this thesis. This needs to be always taken into account when drawing conclusions. Due to the nature of the case study, those conclusions can be hardy confirmed due the uniqueness of the actions performed and the resulting reactions. Whether the case study approach really turns out to be appropriate will be shortly reconsidered after the technical problem has been explained (cf. "Reevaluation of Case Study Preconditions"4.3).

Although the case study seems to be the most obvious choice, other possibilities exist. Easterbrook et al. [6] defines a research method called *Action Research*. Although considered contentious throughout scientific researchers, it is described as to "Solve real-world problem while simultaneously studying the experience of solving the problem". And that fits the nature of this research project quite well. The key criteria for judging the quality of an action research is composed of two aspects: First, the problem must be authentic, which it is, due to the fact that it solves a real world problem. Second, there needs to be authentic knowledge outcomes for the participants. The success or failure of the feature implementation should fulfill this requirement.

The resulting research method which will be used in this thesis will be a so called *Mixed-Method Approach*[6], consisting of a case study combined with action research. Combining a more scientific method with a practical component seems to be the most appropriate here.

4. Solving a Real-World Problem

In order to accomplish this case study from start to end, we need an appropriate real world example. One that has the potential to represent a real problem which might as well be pursued by a real word company. It needs to solve a technical problem or to implement a new feature. On the one hand, the problem or feature must not be specifically designed to fit a research project like this. Simultaneously, it needs to solve a problem or to implement a feature which is obvious and can be easily understood as a problem which needs a solution and thus is most likely to be accepted by the open source community. On the other hand, it must not be too complex, so that it is theoretically solvable in the given limited time frame.

In this thesis, the chosen feature focuses on *dock stations*. A dock station is a separate device designed to extend and improve the hardware limitations of mobile devices such as laptops, but also of smartphones or tablet computers. All of them are specialized computers which in many cases are limited to the hardware components a manufacturer includes at shipment. Having the possibility for changing the hard disk or expanding the laptop via device slots is quite common, yet still limited. This is where dock stations come handy. They can be mechanically attached to the laptop, extending it with additional connections like USB or video, adding expansion slots for PCMCIA devices or hard disks, and can have built-in devices like more powerful graphics adapters or sound cards.

From an operating system perspective, supporting dock stations in a general way is most often solved with the *Advanced Computer Programming Interface* ACPI [1]. The ACPI specification "provides an open standard for device configuration and power management by the operating system"[16]. Via ACPI, the operating system is able to logically attach or detach a dock station device to a computer. The respective ACPI methods are *DCK* for the connection establishment and *EJECT* for the removal. In most cases, the devices contained in the dock station are then initialized and managed via the usual operating system device drivers.

There are also those devices, which can be dynamically plugged into a dock station with an appropriate slot. Very often, those devices are additional batteries or hard disks, to extend the runtime of a mobile device or to expand the storage capacity.

When targeting the personal computer market, operating systems need to support those special devices more extensively than just making the storage or the battery capacity available to the system. The user wants to and needs to interact with those devices. For instance, having an additional battery attached to the system, the user likes to know the remaining capacity, the aggregated total capacity or the time left tak-

ing all available batteries into account. For storage devices, which are often used as sole data or backup additions, even more support is anticipated. Nearly always when storage is involved, a file system which contains and manages the data is in place. Those file systems have special limitations and certain requirements when it comes down to data integrity. Usually you are not allowed to suddenly unplug a hard disk which is still visible and attached to the software stack. File systems have special precautions to prevent data corruption in those cases, however, the risk for damaged data rises. And even if the file system always handles such device removals properly, cached data in the operating system may not have been flushed to the disk when the unplugging occurs. Also, poorly written software might not perfectly deal with a suddenly disappearing storage device. All those reasons make it necessary to tell the operating system and the user about an imminent device removal before it actually occurs. In GNU/Linux based operating systems, this is performed by a user action called *Safely Remove Device*. Similar mechanisms exist in *Microsoft Windows* or *Apple Mac OS X*.

And that is where the specific characteristic of dock stations have to be taken into account. Usually a device like a hard disk is directly attached to a laptop via USB, SATA or another connection possibility. Not so with devices which are attached to a dock stations. Those devices directly depend on the presence and proper connection of the dock station to the supporting computer device. Once the dock stations is removed, at the same time the other device disappears. This obstacle makes it necessary to make the operating system, and last but not least, the user, aware of this direct link. This is most often done via graphical notifications and optional possibilities to manually eject or remove a device.

For this to accomplish, it is mandatory for the operating system to be aware of the fact that the hotpluggable device is directly bound to the dock station or any miscellaneous expansion slot. This might sounds self-evident, however, still misses a proper implementation within the Linux kernel and projects up the stack like the desktop environments and possible middle ware components trying to tie those two levels together.

4.1. Architecture of Modern GNU/Linux Based Operating Systems

In modern GNU/Linux distributions, you find a three layer model. At the bottom, there is the actual operating system, the Linux kernel. For communication with userland[1] applications, it exports a virtual file system called *sysfs*. Userland applications make use of that filesystem to get kernel internal information about devices, system state or the like. The current valid state, for instance attached or detached, of a dock station or

[1] "Userland usually refers to the various programs and libraries that the operating system uses to interact with the kernel: software that performs input/output, manipulates file system objects, etc." [15]

hotpluggable device needs to be exported via this *sysfs*. On top of it, you find a kind of system level applications, also called *daemons*. They usually serve to abstract the very technical and low level information provided by the kernel, to recycle them for easy use by higher level applications. Higher level applications are usually programs running in a graphical desktop environment. Those application receive, modify and query kernel, and thus device information via inter process communication (cf. D-BUS[2]) through the system level daemons.

4.2. Required Implementations

In this specific example, the bottom part is the Linux kernel[3], the system level daemon providing information about storage devices is *UDisks*[4] and the higher level application is the *GNOME* desktop environment[5]. The kernel receives information from the device, exports it via *sysfs*, *UDisks* picks it up and forwards it to the *GNOME* desktop. So it is crucial to this feature implementation and submission process to create and submit each change after another. The changes for the Linux kernel need to be done and submitted before the changes for *UDisks*. Same applies for *UDisks* and *GNOME*.

In the end, the responsible desktop application shows an appropriate reaction to the user which can perform the desired action, like ejecting the device or refraining from unplugging the dock station. Two central questions need to be answered with a concrete implementation:

1. "Which devices are in a dock station?"
2. "Which devices are in a laptop bay?"

So the following actions are supposed to be performed:

1. Evaluation of available hardware and the technical realization of the involved dock station support.
2. Extending the ACPI dock station driver within the Linux kernel to be able to logically assign a hotpluggable device, such as a hard disk, to a bay device slot or to a dock station.
3. Once the previous two preconditions are fulfilled, the middle ware of the GNU/Linux based operating systems needs to be inspected. Higher levels like the desktop environments need a proper way of getting the device information, for instance if a hard disk is actually attached to a dock station or not. This service will provided by *UDisks*.

[2] `http://www.freedesktop.org/wiki/Software/dbus`
[3] `http://www.kernel.org`
[4] `http://www.freedesktop.org/wiki/Software/udisks`
[5] `http://www.gnome.org`

4. Graphical integration and user interaction with attention to usability within a popular GNU/Linux desktop environment. At this place, the *GNOME* desktop environment version 3.0 needs to be modified.

4.3. Reevaluation of the Case Study Preconditions

The described enhancements which have to be performed are fitting the preconditions for the case study handled in this research project pretty well. To be recalled, those preconditions where:

1. Solving a real world problem which might also be taken as an industry goal for a company trying to put a new device on the market.
2. It solves a problem or in this specific case, it implements a new feature.
3. Manageable and doable within the given limited time frame.
4. It is a feature that obviously needs a solution and thus will be most likely accepted by the open source community.

5. Formalities of the Patch Submission Process

Open Source software is different. At least when compared to traditional proprietary, closed source, company driven development models. Although most large open source projects follow the bazaar style[13] of handling their code base, each project has its own ways of communication, maintainer and contributor interaction, code quality requirements, coding style and so on. So before creating code which needs to be made available to the different projects, a few preconditions need to be considered for each of them. Those basically are:

- **Source Code Availability and Access:** There are a bunch of different tools and methods of how to handle source code:

  > "Revision control, also known as version control and source control (and an aspect of software configuration management), is the management of changes to documents, computer programs, large web sites, and other collections of information."[18]

  In the open source area, different revision control systems are used: *CVS*, *SVN* or *GIT*[1], just to mention a few. For instance, while *CVS* and *SVN* rely on a single code location, *GIT* uses a decentralized approach where every developer could use his own code tree.

- **Release Cycle:** One important aspect, especially for the subsequent chapter about defining the proper development schedule (cf. 6), is the **Release Cycle** to deal with. In this context, the release cycle is the period of time from one officially released version, until the next is made publicly available. It can be fixed, say monthly or yearly, or variable, depending on the software stability or bug status.

- **Submission Channel:** When submitting new code or code changes to an existing open source project, a lot of different submission channels are possible. Some project maintainers require the respective patches to be posted to some bug tracking system like Bugzilla[2] or made available via a mailing list which enforces preliminary review. In turn, others like to see code changes made available via

[1] `http://git-scm.com/`
[2] `http://www.bugzilla.org`

local repositories like *GIT* or by simple email submission. The appropriate channel is one of the keystones of an successful submission strategy. If the responsible maintainer needs to perform some circumstantial task to just have a look at the code changes or is not even aware of a change request in the first place, the chances for success significantly decrease. So this needs to be individually be taken care of for each involved project.

- **Coding Style:** One of the key skills a good programmer needs is the ability to adapt his programming style to the project he is targeting. The code base which needs to get extended or modified defines the coding style the existing maintainers and contributors have agreed on and are expecting. Depending on the programming languages, coding styles might vary significantly. When project members are reviewing patches, they expect to be these patches in the same style the existing code is in.

- **Code Quality:** True, code quality should be out of question, always. It should be equally high for every project. However, certain projects might cope better with small mistakes than others. Missing a check or a proper debug output might get accepted in one project and might be the reason for dismissal in another. Setting the same high standards for all written code will give the highest possibility for the changes to be accepted.

- **Patch Format:** Besides code quality and coding style, there is also some kind of meta information some projects require to be submitted alongside the specific code change. This might be meta information like the author's name and email address, other involved people in the patch creation process or statistics of the code changes. This helps projects to track modifications for future reference.

  Throughout the whole open source community, code changes are nearly always created by using a program called *diff*[3], which compares the textual differences between two files. This difference is then submitted to the respective project for inclusion. The project maintainer or an respectively entitled member applies the code change via a program called *patch*[4]. Besides the advantage that the submissions become very small in contrary to a full source code submission, there is another important reason for sending such "diffs": The review process becomes a lot more easy, because you do not have to review the source code as a whole, but can solely look at the changes. Imagine a one line code change in a file of 5000 lines of code. This would not be manageable without a tool chain like *diff* and *patch*.

  Another important aspect is the patch format, especially the logical separation of changes. Some projects might prefer one big patch containing all the involved

[3] `http://www.gnu.org/software/diffutils/`
[4] `http://www.gnu.org/software/diffutils/`

changes. Another might like to see a new feature to get in via small conceptional separated patches which can be reviewed individually. Thus, how the code changes should be submitted heavily depends on the project requirements.

- **Process of Approval:** When patches are submitted via an appropriate submission channel, the usual sequence is that they get reviewed and if the quality, the style and the request is valid, they are approved. There are different ways how this approval might be performed, though. It might happen implicitly, by the changes just getting committed to the proper code repository. Or it might happen with direct actions done by the responsible project members, by giving so called acknowledgements (abbrev.: *ACK*). Individual maintainers of specific areas of the project or code are signalling their approval via mails or by other means which brings another member with rights to perform code changes to actually apply the changes. However, for the patch submitter, only the point in time when the patch got accepted is relevant, not especially the process.

Due to the nature of open source projects and their diversity when it comes down to member interaction, source code management and approval mechanisms, each of the above items need to be considered independently for each related project. So this information is indispensable when it comes to the feature implementation and patch creation.

A lot has been written about these items, however, not too much from a scientific perspective. Most documents are either provided by people directly involved in the projects, such as maintainers and contributors, or are part of the project documentation (cf. 2). The characteristic all those approaches have in common is the fact, that they try to provide a guideline for potential new contributors to prevent possible mistakes which can be repeatedly seen. There is a certain similarity to so called *Frequently Asked Questions (FAQ)*, just in a technical manner. Not all projects have these kind of guidelines though, some have more, some less.

For the specific feature which needs to be implemented within this research thesis, three different projects are expected to be touched. Those are the Linux Kernel[5], a middleware project called UDisks[6] and the *GNOME* desktop environment[7]. The most detailed project documentation, just because it is the largest and serves as the base for all GNU/Linux based distributions, is available for the Linux kernel.

[5] `http://www.kernel.org`
[6] `http://www.freedesktop.org/wiki/Software/udisks`
[7] `http://www.gnome.org`

5.1. The Linux Kernel

The most comprehensive part of kernel documentation is located at the source tree[8] of the code repository itself. Most relevant files for this project are *SubmittingPatches*[4], *ManagementStyle*[3] and *MAINTAINERS*.

Before going into details, the Linux kernel development structure needs to be described briefly. The Linux kernel code and its member affiliations are basically separated into so called subsystems. There are a couple of them, like storage, power management, video, cryptography, input devices or sound. Each of those subsystems has one or more maintainers, also called *kernelmanagers*[3] or *subsystem maintainers*. Each of those kernel managers most often have a own source tree where they manage and track the code changes concerning their specific subsystem. The lead maintainer is Linus Torvalds himself. Every code change needs to be committed by him before it will end up in new official kernel release on *kernel.org*. However, the individual code reviews, approvals and contributor communication for specialized areas are handled by the respective subsystem maintainers and specialists working in these areas. The primary communication within the kernel project is happening on various mailing lists, at least one for each subsystem. Whenever a patch is sent to such a list, it needs to be reviewed. Either by the subsystem maintainer himself or by a reputable community member working in this area. Those are signalling either their acknowledgement, which should result in the code changes being committed to the according source tree, or they are signalling their disagreement. This might happen because of various reasons such as code quality, coding style, functionality or design flaws.

The Linux kernel has a variable **Release Cycle**. Fixed release deadlines cannot be expected. This can be exemplified by the following citation by Andrew Morton posted to the Linux development list (LKML[9]):

> "Nobody knows when a kernel will be released, because it's released according to perceived bug status, not according to a preconceived timeline."

Taking the previous 12 releases into account, it has an average release cycle of 79 days, ranging from 65 days (kernel 3.0.0) to 93 days (3.1.0). Table 5.1 shows all the relevant data.

The time frame from one release to the next can be divided into different time slots. Immediately after a new kernel version is released, a so called merge window opens for two weeks. This is the time Linus Torvalds accepts new code coming from individuals, the subsystem maintainers or other source trees. Especially comprehensive code changes like support for new devices or new features get accepted in this phase. Once the merge window closes, the first release candidate (RC) is tagged and made public. Starting from that point in time, only minor code changes such as coding style fixes, bug fixes

[8] `http://git.kernel.org/`
[9] `https://lkml.org/`

Kernel Version	Release Date	Days of Development
2.6.30	2009-06-10	79
2.6.31	2009-09-09	91
2.6.32	2009-12-03	85
2.6.33	2010-02-24	83
2.6.34	2010-05-16	81
2.6.35	2010-08-01	77
2.6.36	2010-10-20	80
2.6.37	2011-01-05	77
2.6.38	2011-03-15	69
2.6.39	2011-05-19	65
3.0.0	2011-07-23	65
3.1.0	2011-10-24	93
3.2.0	2012-01-05	73

Table 5.1.: Kernel Releases Since July 2009 [2]

or isolated modifications not touching the main areas are accepted. Until the kernel version is final, subsequent release candidates are tagged until the final incarnation is considered to be ready.

However, during the whole time, the subsystem maintainers collect all kinds of new code. They often manage different source trees, one for the kernel currently worked one and one for upcoming versions. This way, they can accept extensive changes without endangering the quality of the release candidates. The changes targeted for future releases can be considered "queued" until the next merge window opens. So it is possible to send patches whenever they are ready, although it is not sure the they will find a lot consideration when posted outside of the merge window.

For further reference, the file *ManagementStyle*[3] has a lot of details on this management process.

SubmittingPatches[4] can be considered a very detailed document for providing a guideline for new contributors. It is titled "How to Get Your Change Into the Linux Kernel" and is summarized as follows:

> "For a person or company who wishes to submit a change to the Linux kernel, the process can sometimes be daunting if you're not familiar with "the system." This text is a collection of suggestions which can greatly increase the chances of your change being accepted."[4]

It is the central starting point for answering the mentioned preconditions stated above. For the **Submission Channel**, the document clearly states the various mailing lists[10]

[10] `http://vger.kernel.org/vger-lists.html`

as primary targets. *SubmittingPatches* refers to the file *MAINTAINERS* which has the detailed subsystem meta information. For the concrete project handled here, the relevant mailing lists are *linux-acpi*[11] where Len Brown is in charge of approvals and *linux-ide*[12], for which Jeff Garzik is the responsible kernel manager. Because of the fact that the new feature is spawning two different subsystems (ACPI and SATA), the **Process of Approval** is a little more complicated here. Both maintainers need to signal their agreement before the according code changes are accepted and can go their way into the main tree.

SubmittingPatches also provides clear guidance on the **Patch Format**, how they need to look like and which meta information needs to be sent along with it. An example of a patch mail fulfilling the requirements can be found in the appendix (cf. A.1.1). Chapter 7.1.1 contains an exemplary step by step guidance on how to create, adapt and send a patch for inclusion into the Linux kernel.

The file also has some minor guidelines for the **Coding Style**. It rather refers to the file *CodingStyle* for more detailed information on this topic.

The **Code Quality** is not mentioned explicitly, however, can be considered quite sophisticated. This is also depending on the concrete maintainer in charge, although in the end, all changes need to pass Linus Torvalds' final judgment.

5.2. UDisks

After looking at the Linux kernel as one of the most active, largest and important open source projects ever existed, attention needs to be paid to a smaller projects, *UDisks*. It is a middle ware program which basic functionality is to abstract and manage storage devices within a computer system. It is located between the kernel and a possible graphical desktop environment. Its main purpose is to abstract storage devices (e.g. hard disks and flash storage) and their functionality by providing methods for querying and manipulating them. Those methods can be easily used by higher levels like graphical desktop environments via inter process communication (cf. D-BUS[13]).

UDisks is maintained and developed essentially by a single person. Also because of the project size, this seems to make the patch submission process more straightforward. For instance, **Patch Approval** is basically done by this single person.

Furthermore, a file called *HACKING* is located in the source code repository[14]. It contains basic information about how to get involved and if you do, how the patches should look like. It states that **Coding Style** should be based on the existing source code and how the canonical **Patch Format** should look like. The web site[7] shows that

[11] linux-acpi@vger.kernel.org
[12] linux-ide@vger.kernel.org
[13] `http://www.freedesktop.org/wiki/Software/dbus`
[14] `http://cgit.freedesktop.org/udisks/tree/`

development mailing list called *devkit-devel*[15] should be used as **Submission Channel** for features and patches. *GIT* is used as **Source Code** revision control system where the code is publicly available to everyone. The **Release Cycle** is not fixed, depending on distribution releases, acuteness of bugs such as security relevant fixes and feature status. Changes might be refused directly ahead of an upcoming release, but should be generally accepted at any time.

5.3. The GNOME Desktop Environment

The second large open source project this thesis has to deal with is the *GNOME* Desktop Environment. Quoting Wikipedia:

> "GNOME [...] is a desktop environment and graphical user interface that runs on top of a computer operating system. It is composed entirely of free and open source software. It is an international project that includes creating software development frameworks, selecting application software for the desktop, and working on the programs that manage application launching, file handling, and window and task management." [17]

In contrary to the rather meager information provided for the development process of UDisks, *GNOME* is more comprehensive in this regard. The base for **Submission Channels** for both previous mentioned projects were mailing lists. Not so with *GNOME*. Within the *GNOME Love Project*[16], which tries to explicitly provide help for new contributions, a sub page deals with the matter of *Submitting Patches*[17]. It states: "Send the patch to the project by attaching it to the relevant bug report in GNOME Bugzilla"[18]. That is the same place where new patches are reviewed and **approved**. The same web site also has information about **Coding Style**, which should match what is already there, and about the **Patch Format**, which is similar to the one which can be found within the Linux kernel. Once again, *GIT* is used as source code repository and its **Source Code** is publicly available.

The *GNOME* project has a quite fixed road map and **Release Cycle**[18]. Current development is active for version 3.4 which is planned to be released on 28 March 2012. However, from a contributor point of view, it does not really matter. As soon as the changes are ready, they are submitted. And due to the fact that *GNOME* is the last project which will be touched, the changes will turn up in the next release.

[15] `http://lists.freedesktop.org/mailman/listinfo/devkit-devel`
[16] `http://live.gnome.org/GnomeLove`
[17] `http://live.gnome.org/GnomeLove/SubmittingPatches`
[18] `https://live.gnome.org/ThreePointThree`

6. Definition of a Submisson Strategy

Three different project, each with its own development process. Sure, similarities exist for all of them, like the source code management system or the way coding style is enforced. However, all are open source projects managed by individuals and not driven by a company aligning a solid road map or schedule. Having three different project depending on each other, defining a concrete and fixed schedule for this thesis is close to impossible.

It gets even more complicated when a new feature is not confined to one single project, but touches different open source efforts. This makes feature development quite unpredictable, especially for companies. Trying to ship a new hardware device to the market and not knowing when the according software drivers will be ready is one of the big obstacles when dealing with open source software designed for GNU/Linux based operating systems. However, this ambiguity cannot always be prevented, even outside the open source community. Development resources are often bound to budget, which in turn is limited, both by size and time.

So when trying to define a schedule or road map, the circumstances need to be explicitly considered, even more than within traditional feature development cycles.

This research project tries to answer the question, if it is possible to circumvent those obstacles and to prove or fail the effort of integrating a new feature into the whole GNU/Linux stack, from the top (desktop) to the bottom (OS/kernel). All this confided to a predefined time frame. Start of the project will be November 15th 2011, end will be targeted for end of February 2012. This gives three and a half month to finish all required work. Three month was the original target, however, got extended by the circumstance that around Christmas and New Year, development and reviewer efforts in the open source community are often reduced.

6.1. Stage 1 - Development and Submission of Linux Kernel Parts

The further the required action is in the future, the fuzzier a prediction can be. Luckily the Linux kernel is the first projects that needs attention, which in turn is the most complex. Having described the exact management process in chapter 5.1, aligning the strategy accordingly is the next logical step.

The last kernel release was on October 24th. Taking an average release cycle of 79 days (cf. 5.1), the next merge window could open around 11th January, 2012. If all

Milestone	Date
Project Start / Start with kernel work	2011-11-15
Initial Kernel Submission	2011-12-11
Kernel Merge Window Opens	2012-01-11
Kernel Merge Window Closes	2012-01-25
Submission for *UDisks*	2012-02-01
Submission for *GNOME*	2012-02-15
Project End	2012-02-29

Table 6.1.: Predicted Submission Schedule

the work should be finished by end of February 2012, all kernel changes need to be submitted, reviewed and committed to at least one of the subsystem maintainers source code trees by the moment the merge window opens. If this time frame is missed, the next but one kernel release needs to be waited for, which will fail the initial plan. So submitting the patches as early as possible is crucial to the success of this project.

So the plan is to do the first submission of the patch set by 11th December. This will give the kernel developers a whole month for the review and acceptance, including bug fixing and possible resubmissions. In parallel and during idle times, when patches have been submitted but feedback is outstanding, work on the userland parts can be done.

During the bi-weekly merge window, predicted start on 11th January, the responsible subsystem maintainer will send his modifications, including the changes related to this thesis, for inclusion into the main Linux tree so Linus Torvalds can pick them up (cf. 6.1).

6.2. Stage 2 - Development and Submission of Userland Parts

Once all needed changes are in the main Linux tree, the corner stone has been set. It is the base for the upcoming work which is entirely located in userland. Ideally, the moment the merge window closes, the work related to the first project, *UDisks*, has already been finished, or is at least near to completion. So within the following week, until 1st February, 2012, the *UDisks* changes can be officially submitted. Accordingly, the *GNOME* changes which have the previous changes as a dependency, can be finished and sent for inclusion into the *GNOME* project within the subsequent two weeks (cf. 6.1). This leaves a buffer of another 14 days if something goes wrong or delayed. Finally, the project should be officially finished on 29th February, 2012.

7. Initial Feature Design and Implementation

After an appropriate submission strategy has been aligned, the concrete feature design and implementation starts. Due to the nature of the open source projects to deal with, especially the Linux kernel, this chapter cannot be finished conclusively. It will rather include the initial design and implementation, but skips further modifications, change requests or bug fixes. The continuation will then be done by the chapter about the actual submission process (cf. 8). Both chapters will be mixed, because the received feedback will inevitable lead to coming back to the practical coding work. To requests on coding style, interface definitions or overall design must be paid attention and acted accordingly. Resubmissions which will get back and forth will be the logical consequence.

7.1. Kernel Space

The first thing to do, which is one of the characteristics of open source development, is to check if the feature to be implemented has already been done or at least tried before. While searching the mailing list archives for linux-acpi[1], a thread[2] related to the targeted feature could be found. It is a series of five patches, promising to implement exactly what is tried to accomplish here. It contains the following patches:

```
[PATCH 1/5] scsi: Export scsi_bus_type
[PATCH 2/5] libata: Bind the Linux device tree to the ACPI device tree
[PATCH 3/5] libata: Migrate ACPI code over to new bindings
[PATCH 4/5] acpi: Add support for linking docks to the objects they contain
[PATCH 5/5] libata: Add links between removable devices and docks
```

As can be seen, four of five patches are targeted for the SATA subsystem, namely the patches tagged with “scsi” and “libata”. The fourth patch is targeted for the *ACPI* subsystem.

Those patches have not received any feedback since September 2010 and the author did not, for whatever reasons, enforce them. They have been based on a kernel code base between version 2.6.35 and 2.6.36. Understandably, because a whole year has passed

[1] http://www.spinics.net/lists/linux-acpi/
[2] http://comments.gmane.org/gmane.linux.acpi.devel/47378

since then, they did not apply cleanly to the current code base. After fixing them to apply, a hardware test unveiled that they even do not work as expected. Because there seems to be no obvious error, further debugging with real hardware was needed. After a lot of debugging, two problems have been identified. One in the SATA and one in the ACPI subsystem. After adequate testing, the changes are ready to be packed into concrete patches for submission.

7.1.1. Patch Set Creation: Iteration One

After the technical implementation problems are solved, the concrete patches for submission have to be created. The following sequence is heavily oriented on the guideline found in the documentation file *SubmittingPatches*[4] and can be considered standard procedure:

1. Check out the relevant main Linux development tree[3] to get the source code of the Linux kernel.
2. Perform the necessary changes to the source code files. In this specific case, two logical separate changes, one for the SATA and one for the ACPI subsystem, are needed. This will give the patches a higher change of being accepted, for multiple reasons:
   a) The changes are easier to review, because they do not span multiple subsystems or logical code paths.
   b) Each individual maintainer can give their approval independently for each subsystem.
   c) If there are possible future problems, it is easier to only revert the responsible patch instead of reverting all changes at once.
3. Make sure the changes fulfill the coding style requirements.
4. For each of them, create a "unified diff"[4] output.
5. Create an appropriate patch description. One one-line statement, which will become an unique identifier for the patch. Then create a multiple line description, containing a detailed explanation of the problem the patch is solving, what it is doing and further optional references. Both descriptions will become part of the commit message in the main Linux development tree of Linus Torvalds.
6. Select the proper E-mail destination. Judging from the corresponding *MAINTAINERS* file contained within the Linux kernel documentation, this is *linux-acpi@vger.kernel.org* for the ACPI subsystem and *linux-scsi@vger.kernel.org* for

[3] git clone `https://github.com/torvalds/linux.git`
[4] diff -up, -u: "unified diff format", -p: "show relevant function the change is in"

the SATA subsystem. The created patch set need to be sent to both destinations simultaneously, because it contains changes spawning the two subsystems.

7. Create the proper list of additional recipients, who will be put into the carbon copy (CC) list of the mails. Obviously this is the author of the original five patches found in the mail archives. Other additional recipients do not seem to be necessary.

8. Prepare and send the E-mails. At this point, a quotation from *SubmittingPatches*[4] is supposed to describe the mail content.

```
The canonical patch subject line is:

    Subject: [PATCH 001/123] subsystem: summary phrase

The canonical patch message body contains the following:

  - A "from" line specifying the patch author.

  - An empty line.

  - The body of the explanation, which will be copied to the
    permanent changelog to describe this patch.

  - The "Signed-off-by:" lines, described above, which will
    also go in the changelog.

  - A marker line containing simply "---".

  - Any additional comments not suitable for the changelog.

  - The actual patch (diff output).
```

Further reading about the concrete patch and E-mail format can be provided by "The Perfect Patch"[12] from Andrew Morton.

As a result, a number of 8 patch mails are created (cf. A.1.2). The first mail is a summary of the patch set, what it does and what it is intended for:

```
[PATCH 0/7] acpi/libata: Express dependencies for devices on dock stations and bays
```

To quote in full:

```
Patches 1 through 5 are just a refresh of the patches from Matthew
Garrett sent to this list in September 2010 [1]:
```

```
 [PATCH 1/7] scsi: Export scsi_bus_type
 [PATCH 2/7] libata: Bind the Linux device tree to the ACPI device tree
 [PATCH 3/7] libata: Migrate ACPI code over to new bindings
 [PATCH 4/7] acpi: Add support for linking docks to the objects they contain
 [PATCH 5/7] libata: Add links between removable devices and docks

Patches 6 and 7 make the patches actually work on my test hardware
(Thinkpad x60/Thingpad T60) by fixing some minor issues.

 [PATCH 6/7] libata: Generate and pass correct acpi handles
 [PATCH 7/7] acpi: Prevent duplicate hotplug device registration on dock stations

Regards,
 Holger

[1] http://comments.gmane.org/gmane.linux.acpi.devel/47378
```

A finished, exemplary mail body looks like that:

```
Fix ACPI handle generation for device handles and pass the correct
handles to the dock driver.

Signed-off-by: Holger Macht <holger@homac.de>
---
 libata-acpi.c |   10 ++++-------
 1 file changed, 3 insertions(+), 7 deletions(-)
[diff output truncated]
```

The complete content of the first iteration of patches can be found in appendix A.1.1.

7.2. Userland: UDisks and GNOME

Before moving on to the actual patch submission process, the userland projects (*UDisks* and *GNOME*) need some consideration. In concrete, the *UDisks* project need to be extended to represent the required changes. It needs to pick up the information regarding dock stations provided by the kernel *sysfs* interface and is intended to pass them on to desktop applications via inter process communication (D-Bus). Then, the *GNOME* desktops needs an appropriate representation for the user. Both will be, as time permits, taken a look at after the first submission of kernel patches, when feedback has been received and in turn is supposed to be incorporated.

8. Feature Submission Process

In short, the submission process is summarized in the following table:

Milestone	Date
Project start / Start with kernel work	2011-11-15
Initial kernel submission	2011-12-06
Merge window for 3.3.0 opens, reminder mail sent	2012-01-05
ACPI maintainer sends pull request to Linus Torvalds	2012-01-17
New remainder for ACPI subsystem maintainer	
ACK from ACPI subsystem maintainer	2012-01-18
Initial feedback for minor issues received	
ACK for valid patches from SATA subsystem maintainer	
Resubmission 1, adding two new patches	2012-01-20
ACK from SATA Maintainer	2012-01-21
Queue for linux-next	2012-02-09
Community member has problems with the patch set - Fix required	2012-02-18
Additional two patches are accepted	2012-02-21
Project end	2012-02-29

8.1. Initial Submission

The first submission of Linux kernel patches (cf. 8) has been done on 6th December, 2011. As *SubmittingPatches* proposes with "After you have submitted your change, be patient and wait.", it is done. Following the release cycle of the release candidated of the Linux kernel, the release of final 3.2.0 kernel and thus the opening of the next merge window got closer and closer. On 5th January, 2012, kernel 3.2.0 was final and with it, the merge window for 3.3.0 opened. However, without any feedback on the submitted patches, it seemed to be very unlikely that they would be included. Thus, a reminder mail[1] asking the original author and the subsystem maintainers for feedback has been prepared. Again, no feedback. On 17th January, 2012, one day before the merge window closes, Len Brown, the ACPI subsystem maintainer sent a pull request[2] to Linus Torvalds, including the relevant changes from the ACPI subsystem for inclusion in kernel 3.3.0. The patches related to this project were not included, though. Hereupon,

[1] `http://comments.gmane.org/gmane.linux.scsi/71386`
[2] `http://permalink.gmane.org/gmane.linux.kernel/1240185`

another reminder mail[3] was sent, asking if there is a specific reason why the patches have not been considered. The result: An approval from the ACPI subsystem maintainer on 18th January, 2012. However, only for the parts concerning the ACPI subsystem.

8.2. Missing the First Kernel Release Cycle

The parts belonging to the SCSI still waited for review. As already suspected, and because the 18th was supposed to be the day the merge window closes, the approval came too late for a possible inclusion in kernel 3.3.0. However, and after the stone has been set rolling, feedback from the SATA subsystem maintainers did not take long. Minor issues need to get resolved, otherwise the SATA subsystem maintainer, Jeff Garzik, will pick the patches up so they can be queued for kernel 3.4.0 and integration testing can be done in the linux-next tree. To fix the minor issues, a new iteration of patches was required.

8.3. Strategy Adaption

Due to the fact that the first iteration of kernel patches missed the required deadline for kernel version 3.3.0, a strategy adaption was necessary. The patches turned out to be too intrusive to just get them committed to both relevant subsystems. That most likely was the reason for the SATA maintainer pushing them to *linux-next*[4] for integration testing. If no problems arise until the next merge window opens (approx. beginning of march, cf. 5.1). Because of that, primary focus was shifted from touching the whole stack (Kernel, *GNOME*, *UDisks*) to just getting the relevant kernel changes accepted. Although the other goals are still pursued, the top priority lies within the kernel patches, because they make up the base for any further work in this area. For reference, the work done on the *UDisks* projects can be found in the appendix A.2.

8.4. Reacting on Feedback from the Community

After the first important milestone has been missed, focus lies on refreshing the patches according to the feedback received. Of course, this includes repeating all the testing work done before. Furthermore, the underlying code base has changed, from what was available in the development tree during 3.1.0 and 3.2.0, to a version based on 3.2.0, in addition to the changes which got pulled in during the passed merge window. So if unlucky, all the patches need to be rebased according to the modified code base.

As a first change, the patch formerly known as patch 1 of 7 was replaced by a own version:

[3] `http://permalink.gmane.org/gmane.linux.acpi.devel/51756`
[4] A separate development tree used for integration testing

```
[PATCHv2 1/8] scsi: Add wrapper to access and set scsi_bus_type \
                    in struct acpi_bus_type
```

Additionally, it turned out during testing, that the patches did not work anymore as expected. Because no obvious explanation was available, the only remaining reason has to be the modified code base. So something else, not directly related to the patch set must have been changed which affects the new feature. Debugging confirmed that suspicion and an additional patch (cf. A.1.2) was the result:

```
[PATCHv2 8/8] libata: Use correct PCI devices
```

8.4.1. Resubmission 1

On 20th January, 2012, a new patch set[5], containing 8 patches (cf. A.1.2), was sent to the appropriate lists:

```
Patches 2 through 5 are just a refresh of the patches from Matthew
Garrett sent to this list in September 2010 [1]:

Patch 1 is a new patch incorporating the corrections from James
Bottomley. Patch 6, 7 and 8 make the whole patch set actually work on my
test hardware (Thinkpad x60/Thinkpad T60) by fixing minor issues and
compensating changes after the first submission.

All patches now contain the correct Signed-off-by instead of Acked-by
tags.

 [PATCH 1/8] scsi: Add wrapper to access and set scsi_bus_type in struct acpi_bus_type
 [PATCH 2/8] libata: Bind the Linux device tree to the ACPI device tree
 [PATCH 3/8] libata: Migrate ACPI code over to new bindings
 [PATCH 4/8] acpi: Add support for linking docks to the objects they contain
 [PATCH 5/8] libata: Add links between removable devices and docks
 [PATCH 6/8] libata: Generate and pass correct acpi handles
 [PATCH 7/8] acpi: Prevent duplicate hotplug device registration on dock stations
 [PATCH 8/8] libata: Use correct PCI devices

Regards,
 Holger

[1] http://comments.gmane.org/gmane.linux.acpi.devel/47378
```

[5] http://thread.gmane.org/gmane.linux.acpi.devel/51785

After more than two weeks, and after another interested community member asked for the status of these patches, the SATA subsystem maintainer finally pushed the patches to *linux-next*[6] for integration testing on 9th February, 2012. This usually means, that if no problem related to the patches comes up, they will be automatically considered for inclusion during the upcoming merge window.

8.5. Reacting on Feedback Cont.

8.5.1. Fixing a Boot Problem

While working on the userland parts and waiting for the kernel changes to finally get merged into Linus Torvalds tree, a community member complained on the main Linux kernel development list[7] about boot problems while testing the latest *linux-next* tree. The title of the thread is "linux-next: dock_link_device is oopsy", and seemed suspiciously related to the changes performed in this project. Some debugging proved that assumption. After some back and forth, testing, writing and rewriting a possible patch, yet another patch(cf. A.1.3) on top of the already existing patch series was the outcome:

```
[PATCH] dock: fix bootup oops and other dock_link breakage

dock_link_device() and dock_unlink_device() should bail out early
to avoid oops on zero-length kmalloc() when dock_station_count is 0.

But isn't there an off-by-one in that kmalloc() length anyway?
An extra NULL appended at the end suggests so.

Rework the ordering with gotos on failure to fix several issues.

And presumably dock_unlink_device() should be presenting the same
interface as dock_link_device(), with NULL returned when none found.

Signed-off-by: Hugh Dickins <hughd <at> google.com>
---
[diff output truncated]
```

8.5.2. Fixing a Compilation Error

While debugging the previously mentioned problem, a minor issue and rather cosmetic problem came up. It required yet another patch (cf. A.1.3):

[6] http://article.gmane.org/gmane.linux.acpi.devel/51950
[7] http://comments.gmane.org/gmane.linux.kernel/1254952

```
[PATCH] acpi: Fix compiler error when setting CONFIG_ACPI_DOCK=n

When compiling with CONFIG_ACPI_DOCK=n,
is_registered_hotplug_dock_device() needs to be defined

Signed-off-by: Holger Macht <holger <at> homac.de>
---
[diff output truncated]
```

8.6. Drawing a Final Stroke

Both additional patches mentioned in the previous section where picked up and accepted on 21th February, 2012. Nothing related happened anymore until the end of February. So those two patches can be considered the final actions performed within this research project, which in turn can be marked as completed. Due to the nature of the Linux development process, there is nothing which could be done anymore, until either new problems arise or the feature gets released along with the upcoming official kernel release. Sure, contributor responsibility would not stop at this point. After the submission process, the time of maintenance begins. However, this is not subject of this thesis. More interesting in this regard is the recapitulation of the outlined strategy, the actual submission process, what could have been done better or not.

9. Recapitulation

As a short executive summary, the case study did not work out as expected, and failed, all in all. However, only in regard to the timeline predicted. The new feature was accepted in the largest project involved. It just was delayed by unresponsiveness of the responsible community members and by technical obstacles. Nothing unusual when dealing with software design and implementation. All in all: The feature found acceptance in the Linux kernel project. And if all goes well, it will be in the main code base after the next merge window and will subsequently show up in kernel 3.4.0. If that happens, it will become an inherent part of many GNU/Linux based operating systems. However, the schedule outlined for this case study was undoubtedly missed.

9.1. Schedule Deviance

In terms of the previously aligned schedule, it requires an in-depth look. The initial design and implementation worked out as expected. The required patches were ready by beginning of December 2012. In fact, the first milestone, the initial kernel submission has been beaten by a number of five days. However, this did not turn out as an advantage. The goal was to have the patches submitted, reviewed, possibly fixed and accepted by the respective subsystem maintainer the moment the new merge window opened. This window did open, six days after what has been predicted. However, within a whole month, the patches did not receive any further recognition, what unfortunately lead to missing the merge window. This turned out to be the first and future omnipresent violation of what has been predicted in the submission schedule. It is fatal for every kind of schedule when the first important deadlines are missed. The earlier the deviation from a schedule, the more problematic to cope with.

9.2. Impact of Missing the Deadlines

At this point in time, it was clear that the first and possible most intrusive part of patches will miss the required 3.3.0 kernel release. This lead to a huge delay and caused a blockage of the remaining user space modifications and thus, the remaining schedule. The remaining days, the focus was set to getting the kernel changes officially accepted, which in turn implied to defer the userland implementation. The reason for that is primarily that you will most likely not get any code changes into a project, whose

underlying interface support is not officially accepted nor widely used. Another reason is that the moment the first kernel review came in, development resources were bound to fixing the remaining issues instead of concentrating on other parts. As the kernel modifications again needed some attention by the end of February, it was undoubtedly too late to engage other projects.

9.3. Localizing the Problems

Although it is clear what went wrong from a timely manner, the root causes have not been identified yet. They can be considered manifold and whether they can really be made responsible often remains subject to speculation. The following sections are rather a *fishing for root causes* instead of reciting proven facts. The causes could be categorized into three different areas: Problems caused by personal matters, by the nature of open source or by the individual project at hand.

9.3.1. Caused by Personal Matters

In the area of personal matters, one has to mention the fact that the author of the patches has no big reputation in the area of kernel development. He occasionally submitted minor bug fixes or enhancements, yet not to the extent this thesis deals with. In an ideal world, this should not play a big role, however, in reality it does. So this might be one of the reason why the patch set submission did not get an immediate review. Not knowing the contributor or his skills can cause reluctance by the responsible kernel maintainers. One speculative reason might be, that they cannot be sure about the code quality without having a deeper look. This might lead to putting the request for review to the end of their queue. A very small patch fixing a bug would eventually be accepted silently, not so here.

9.3.2. Caused by the Nature of Open Source

The nature of open source software comes along with some characteristic preconditions. Strictly speaking, nobody is obliged to review the code you submitted. Many people are not paid for doing so, so they pick what they are interested in. If the problem at hand is not of broader interest, nobody feels obliged to review the code. Sure, sooner or later it will happen, as could seen in this case study.

9.3.3. Caused by the Individual Project

Although every subsystem within the Linux kernel has specific maintainer, this person does not necessarily be responsible for the review of every bit of code. Every subsystem has a number of good engineers, often working on exclusive topics. So if nobody really

feels responsible, the code does not get reviewed. Maybe the qualified developer is currently not available or has responsibilities with higher priority. Of course, this would also apply to other large open source projects, but fits particularly well for the Linux kernel development model and structure.

Furthermore, the Linux kernel does not have fixed road map linked with a correlative feature database. Other projects might have the rule to only release when all documented and necessary features are properly implemented and stable. However, not so with the Linux kernel. Everything not accepted and committed by a qualified kernel maintainer by the time the merge window closes will not be shipped. David G. Glance summarized this quite accurate:

> "What functionality goes into a particular version is therefore determined simply by what is accepted during a particular time window."[7]

After trying to identify some of the problems which might have caused the failure of the initial approach, it is time to look at what could have been done better.

9.4. Identified Best Practices

Although the chronology of the concrete submission process cannot be considered smooth, it also can not be called extraordinary. Having an initial patch set submission followed by two resubmissions is kind of standard procedure. Although the case studies result can be considered *failed*, it is not because the feature was not accepted, but rather because of a wrong schedule with exaggerated expectations.

So the first advice which can be given is one which applies to every software project, no matter which development cycle it pursues: Arrange enough time, more than you expect. More lead time, especially for submitting kernel changes, is one of the crucial preconditions for a possible success. Do not target the next kernel release, instead target the next but one.

Furthermore, more than a month passed before the patches received any public review. During those five weeks, the project lied dormant. Andi Kleens advice from "On submitting kernel patches"[8] could have been considered more carefully:

> "Sometimes a submission gets stuck during the submission process. In this case, it is a good idea to just send private mail to the maintainer who is responsible and ask advice on how to proceed with the merge."[8]

This could have been done, however, with some constrains. A lot of the local kernel documentation proposes to be patient, so does the previously mentioned file *Submitting-Patches*. If the change request and its acceptance is of immediate urgency, sending a private mail to one of the subsystem maintainers is one possibility for trying to apply some pressure. However, this could backfire, too. The kernel documentation explicitly

lists some reasons why a patch might not be considered for inclusion. One of them is: "You are being annoying."[4]. So the general rule applies: Be patient, usually you cannot do anything about it.

10. Conclusion

Although this thesis turned out to be specialized on the Linux kernel, a lot can be applied for other open source projects. Most of them might not have the same development model, but similarities exist in all of them. Especially when it comes to code quality, the submission channel or mailing list communication. The Linux kernel development process is just an excellent model of how a large successful open source project can work. So in general, many aspects examined can be transferred to similar open source projects.

When it comes to the outcome of this thesis, one could argue that the project failed and thus companies or new contributors should hold off from investing into open source software development and its corresponding projects. However, that would be too easy. Strictly speaking, only the outlined schedule failed and this was caused by multiple reasons. If all would have went well, with early feedback and acceptance, the schedule could have been fulfilled. The reason this did not come true was simply the fact that the success of the schedule depended on external parameters. Those were the projects which have been dealt with, especially the Linux kernel.

It has, as most open source projects out there, its own rules and processes companies cannot rely on being able to influence in whatever way they like. A major decision, like the inclusion of new code, cannot be enforced. Expecting something like the feature inclusion to work within such a strict time frame is possible, although unpredictable to a certain extend. You have to rely on external project members. At first glance, a company has no means to make this process faster. The only way is to invest, to build up a own contributor base, thus strengthening the own influence by providing development resources. And this can be a tedious process.

This process can be considered worthwhile, though. Although this thesis concentrated on the actual submission process, a contribution to an open source project does not end with the acceptance of the requested changes. After code has been committed to a project's repository, maintenance starts. This is no different with the Linux kernel. Ideally, the contributor takes care of all problems which may arise once his source code has been released with an official kernel. However, if a feature is an inherent part of a big project, the appropriate community might take care of its maintenance. Even if it takes longer to get a feature accepted, as soon as in the kernel, it will be co-maintained by the community and not entirely by the original authors. In the past, companies tried to maintain specific extensions entirely on their own, and quite often miserably failed. Getting the desired changes into the mainline kernel usually means that all the different Linux distributions will pick it up and ship it, automatically. This will in turn extend the user base and can lead to growing development resources. Interested community

members start using the code, have new ideas, find bugs and might finally start to stabilize and improve the corresponding source code by themselves. It would not be the first time a proper feature or idea develops a life of its own nobody would have ever thought of before.

A. Appendix

A.1. Patch Mails for the Linux Kernel

A.1.1. First Iteration of Patches (2011-12-06)

[PATCH 1/7] scsi: Export scsi_bus_type

```
Subject: [PATCH 1/7] scsi: Export scsi_bus_type

From: Matthew Garrett <mjg@redhat.com>

We need scsi_bus_type in order to be able to bind ata devices against
acpi devices. Export it from the scsi core.

Signed-off-by: Matthew Garrett <mjg@redhat.com>
Acked-by: Holger Macht <holger@homac.de>
---
 drivers/scsi/scsi_priv.h |    1 -
 include/scsi/scsi.h      |    2 ++
 2 files changed, 2 insertions(+), 1 deletion(-)

Index: linux/drivers/scsi/scsi_priv.h
===================================================================
--- linux.orig/drivers/scsi/scsi_priv.h
+++ linux/drivers/scsi/scsi_priv.h
@@ -134,7 +134,6 @@ extern int scsi_sysfs_target_initialize(
 extern struct scsi_transport_template blank_transport_template;
 extern void __scsi_remove_device(struct scsi_device *);

-extern struct bus_type scsi_bus_type;
 extern const struct attribute_group *scsi_sysfs_shost_attr_groups[];

 /* scsi_netlink.c */
Index: linux/include/scsi/scsi.h
===================================================================
--- linux.orig/include/scsi/scsi.h
+++ linux/include/scsi/scsi.h
@@ -187,6 +187,8 @@ struct scsi_cmnd;

 #define SCSI_MAX_VARLEN_CDB_SIZE 260

+extern struct bus_type scsi_bus_type;
+
 /* defined in T10 SCSI Primary Commands-2 (SPC2) */
 struct scsi_varlen_cdb_hdr {
        __u8 opcode;            /* opcode always == VARIABLE_LENGTH_CMD */
```

[PATCH 2/7] libata: Bind the Linux device tree to the ACPI device tree

```
Subject: [PATCH 2/7] libata: Bind the Linux device tree to the ACPI device tree

From: Matthew Garrett <mjg@redhat.com>
```

```
We want to be able to express the dependencies between ACPI dock devices
and their children. This requires us to be able to associate the ACPI
device tree and libata devices. This patch uses the generic ACPI glue
framework to do so.

Signed-off-by: Matthew Garrett <mjg@redhat.com>
Acked-by: Holger Macht <holger@homac.de>
---
 acpi/glue.c       |    2
 ata/libata-acpi.c |  118 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 ata/libata-core.c |    3 +
 ata/libata.h      |    4 +
 4 files changed, 127 insertions(+)

Index: linux/drivers/ata/libata-acpi.c
===================================================================
--- linux.orig/drivers/ata/libata-acpi.c
+++ linux/drivers/ata/libata-acpi.c
@@ -47,6 +47,31 @@ static void ata_acpi_clear_gtf(struct at
        dev->gtf_cache = NULL;
 }

+static acpi_handle ap_acpi_handle(struct ata_port *ap)
+{
+       if (ap->flags & ATA_FLAG_ACPI_SATA)
+               return NULL;
+       return DEVICE_ACPI_HANDLE(&ap->scsi_host->shost_gendev);
+}
+
+static acpi_handle dev_acpi_handle(struct ata_device *dev)
+{
+       acpi_integer adr;
+       struct ata_port *ap = dev->link->ap;
+
+       if (dev->sdev)
+               return DEVICE_ACPI_HANDLE(&dev->sdev->sdev_gendev);
+
+       if (ap->flags & ATA_FLAG_ACPI_SATA) {
+               if (!sata_pmp_attached(ap))
+                       adr = SATA_ADR(ap->port_no, NO_PORT_MULT);
+               else
+                       adr = SATA_ADR(ap->port_no, dev->link->pmp);
+               return acpi_get_child(DEVICE_ACPI_HANDLE(ap->host->dev), adr);
+       } else
+               return acpi_get_child(ap_acpi_handle(ap), dev->devno);
+}
+
 /**
  * ata_acpi_associate_sata_port - associate SATA port with ACPI objects
  * @ap: target SATA port
@@ -1018,3 +1043,96 @@ void ata_acpi_on_disable(struct ata_devi
 {
        ata_acpi_clear_gtf(dev);
 }
+
+static int is_pci_ata(struct device *dev)
+{
+       struct pci_dev *pdev;
+
+       if (!is_pci_dev(dev))
+               return 0;
+
+       pdev = to_pci_dev(dev);
+
+       if ((pdev->class >> 8) != PCI_CLASS_STORAGE_SATA &&
+           (pdev->class >> 8) != PCI_CLASS_STORAGE_IDE)
+               return 0;
```

```
+
+       return 1;
+}
+
+static int ata_acpi_bind_host(struct device *dev, int host, acpi_handle *handle)
+{
+       struct Scsi_Host *shost = dev_to_shost(dev);
+       struct ata_port *ap = ata_shost_to_port(shost);
+
+       if (ap->flags & ATA_FLAG_ACPI_SATA)
+               return -ENODEV;
+
+       *handle = acpi_get_child(DEVICE_ACPI_HANDLE(dev->parent), ap->port_no);
+
+       if (!*handle)
+               return -ENODEV;
+
+       return 0;
+}
+
+static int ata_acpi_bind_device(struct device *dev, int channel, int id,
+                               acpi_handle *handle)
+{
+       struct device *host = dev->parent->parent;
+       struct Scsi_Host *shost = dev_to_shost(host);
+       struct ata_port *ap = ata_shost_to_port(shost);
+       struct ata_device *ata_dev;
+
+       if (ap->flags & ATA_FLAG_ACPI_SATA)
+               ata_dev = &ap->link.device[channel];
+       else
+               ata_dev = &ap->link.device[id];
+
+       *handle = dev_acpi_handle(ata_dev);
+
+       if (!*handle)
+               return -ENODEV;
+
+       return 0;
+}
+
+static int ata_acpi_find_device(struct device *dev, acpi_handle *handle)
+{
+       unsigned int host, channel, id, lun;
+
+       if (sscanf(dev_name(dev), "host%u", &host) == 1) {
+               if (!is_pci_ata(dev->parent))
+                       return -ENODEV;
+
+               return ata_acpi_bind_host(dev, host, handle);
+       } else if (sscanf(dev_name(dev), "%d:%d:%d:%d",
+                       &host, &channel, &id, &lun) == 4) {
+               if (!is_pci_ata(dev->parent->parent->parent))
+                       return -ENODEV;
+
+               return ata_acpi_bind_device(dev, channel, id, handle);
+       } else
+               return -ENODEV;
+}
+
+static int ata_acpi_find_dummy(struct device *dev, acpi_handle *handle)
+{
+       return -ENODEV;
+}
+
+static struct acpi_bus_type ata_acpi_bus = {
+       .bus = &scsi_bus_type,
+       .find_bridge = ata_acpi_find_dummy,
```

```
+       .find_device = ata_acpi_find_device,
+};
+
+int ata_acpi_register(void)
+{
+       return register_acpi_bus_type(&ata_acpi_bus);
+}
+
+void ata_acpi_unregister(void)
+{
+       unregister_acpi_bus_type(&ata_acpi_bus);
+}
Index: linux/drivers/ata/libata-core.c
===================================================================
--- linux.orig/drivers/ata/libata-core.c
+++ linux/drivers/ata/libata-core.c
@@ -6442,6 +6442,8 @@ static int __init ata_init(void)

        ata_parse_force_param();

+       ata_acpi_register();
+
        rc = ata_sff_init();
        if (rc) {
                kfree(ata_force_tbl);
@@ -6468,6 +6470,7 @@ static void __exit ata_exit(void)
        ata_release_transport(ata_scsi_transport_template);
        libata_transport_exit();
        ata_sff_exit();
+       ata_acpi_unregister();
        kfree(ata_force_tbl);
 }

Index: linux/drivers/ata/libata.h
===================================================================
--- linux.orig/drivers/ata/libata.h
+++ linux/drivers/ata/libata.h
@@ -117,6 +117,8 @@ extern void ata_acpi_on_resume(struct at
 extern int ata_acpi_on_devcfg(struct ata_device *dev);
 extern void ata_acpi_on_disable(struct ata_device *dev);
 extern void ata_acpi_set_state(struct ata_port *ap, pm_message_t state);
+extern int ata_acpi_register(void);
+extern void ata_acpi_unregister(void);
 #else
 static inline void ata_acpi_associate_sata_port(struct ata_port *ap) { }
 static inline void ata_acpi_associate(struct ata_host *host) { }
@@ -127,6 +129,8 @@ static inline int ata_acpi_on_devcfg(str
 static inline void ata_acpi_on_disable(struct ata_device *dev) { }
 static inline void ata_acpi_set_state(struct ata_port *ap,
                                      pm_message_t state) { }
+static inline int ata_acpi_register(void) { return 0; }
+static void ata_acpi_unregister(void) { }
 #endif

 /* libata-scsi.c */
Index: linux/drivers/acpi/glue.c
===================================================================
--- linux.orig/drivers/acpi/glue.c
+++ linux/drivers/acpi/glue.c
@@ -39,6 +39,7 @@ int register_acpi_bus_type(struct acpi_b
        }
        return -ENODEV;
 }
+EXPORT_SYMBOL(register_acpi_bus_type);

 int unregister_acpi_bus_type(struct acpi_bus_type *type)
 {
@@ -54,6 +55,7 @@ int unregister_acpi_bus_type(struct acpi
```

```
        }
        return -ENODEV;
 }
+EXPORT_SYMBOL(unregister_acpi_bus_type);

 static struct acpi_bus_type *acpi_get_bus_type(struct bus_type *type)
 {
```

[PATCH 3/7] libata: Migrate ACPI code over to new bindings

```
Subject: [PATCH 3/7] libata: Migrate ACPI code over to new bindings

From: Matthew Garrett <mjg@redhat.com>

Now that we have the ability to directly glue the ACPI namespace to the
driver model in libata, we don't need the custom code to handle the same
thing. Remove it and migrate the functions over to the new code.

Signed-off-by: Matthew Garrett <mjg@redhat.com>
Acked-by: Holger Macht <holger@homac.de>
---
 drivers/ata/libata-acpi.c |  166 +++++++++++-------------------------
 drivers/ata/libata-core.c |    3
 drivers/ata/libata-pmp.c  |    4 -
 drivers/ata/libata.h      |    5 -
 drivers/ata/pata_acpi.c   |    4 -
 include/linux/libata.h    |    7 -
 6 files changed, 45 insertions(+), 144 deletions(-)

Index: linux/drivers/ata/libata-acpi.c
===================================================================
--- linux.orig/drivers/ata/libata-acpi.c
+++ linux/drivers/ata/libata-acpi.c
@@ -47,14 +47,29 @@ static void ata_acpi_clear_gtf(struct at
        dev->gtf_cache = NULL;
 }

-static acpi_handle ap_acpi_handle(struct ata_port *ap)
+/**
+ * ata_ap_acpi_handle - provide the acpi_handle for an ata_port
+ * @ap: the acpi_handle returned will correspond to this port
+ *
+ * Returns the acpi_handle for the ACPI namespace object corresponding to
+ * the ata_port passed into the function, or NULL if no such object exists
+ */
+acpi_handle ata_ap_acpi_handle(struct ata_port *ap)
 {
        if (ap->flags & ATA_FLAG_ACPI_SATA)
                return NULL;
        return DEVICE_ACPI_HANDLE(&ap->scsi_host->shost_gendev);
 }
+EXPORT_SYMBOL(ata_ap_acpi_handle);

-static acpi_handle dev_acpi_handle(struct ata_device *dev)
+/**
+ * ata_dev_acpi_handle - provide the acpi_handle for an ata_device
+ * @dev: the acpi_device returned will correspond to this port
+ *
+ * Returns the acpi_handle for the ACPI namespace object corresponding to
+ * the ata_device passed into the function, or NULL if no such object exists
+ */
+acpi_handle ata_dev_acpi_handle(struct ata_device *dev)
 {
        acpi_integer adr;
```

```
        struct ata_port *ap = dev->link->ap;
@@ -69,66 +84,9 @@ static acpi_handle dev_acpi_handle(struc
                        adr = SATA_ADR(ap->port_no, dev->link->pmp);
                return acpi_get_child(DEVICE_ACPI_HANDLE(ap->host->dev), adr);
        } else
-               return acpi_get_child(ap_acpi_handle(ap), dev->devno);
-}
-
-/**
- * ata_acpi_associate_sata_port - associate SATA port with ACPI objects
- * @ap: target SATA port
- *
- * Look up ACPI objects associated with @ap and initialize acpi_handle
- * fields of @ap, the port and devices accordingly.
- *
- * LOCKING:
- * EH context.
- *
- * RETURNS:
- * 0 on success, -errno on failure.
- */
-void ata_acpi_associate_sata_port(struct ata_port *ap)
-{
-       WARN_ON(!(ap->flags & ATA_FLAG_ACPI_SATA));
-
-       if (!sata_pmp_attached(ap)) {
-               u64 adr = SATA_ADR(ap->port_no, NO_PORT_MULT);
-
-               ap->link.device->acpi_handle =
-                       acpi_get_child(ap->host->acpi_handle, adr);
-       } else {
-               struct ata_link *link;
-
-               ap->link.device->acpi_handle = NULL;
-
-               ata_for_each_link(link, ap, EDGE) {
-                       u64 adr = SATA_ADR(ap->port_no, link->pmp);
-
-                       link->device->acpi_handle =
-                               acpi_get_child(ap->host->acpi_handle, adr);
-               }
-       }
-}
-
-static void ata_acpi_associate_ide_port(struct ata_port *ap)
-{
-       int max_devices, i;
-
-       ap->acpi_handle = acpi_get_child(ap->host->acpi_handle, ap->port_no);
-       if (!ap->acpi_handle)
-               return;
-
-       max_devices = 1;
-       if (ap->flags & ATA_FLAG_SLAVE_POSS)
-               max_devices++;
-
-       for (i = 0; i < max_devices; i++) {
-               struct ata_device *dev = &ap->link.device[i];
-
-               dev->acpi_handle = acpi_get_child(ap->acpi_handle, i);
-       }
-
-       if (ata_acpi_gtm(ap, &ap->__acpi_init_gtm) == 0)
-               ap->pflags |= ATA_PFLAG_INIT_GTM_VALID;
+               return acpi_get_child(ata_ap_acpi_handle(ap), dev->devno);
 }
+EXPORT_SYMBOL(ata_dev_acpi_handle);
```

```
 /* @ap and @dev are the same as ata_acpi_handle_hotplug() */
 static void ata_acpi_detach_device(struct ata_port *ap, struct ata_device *dev)
@@ -254,56 +212,6 @@ static const struct acpi_dock_ops ata_ac
 };

 /**
- * ata_acpi_associate - associate ATA host with ACPI objects
- * @host: target ATA host
- *
- * Look up ACPI objects associated with @host and initialize
- * acpi_handle fields of @host, its ports and devices accordingly.
- *
- * LOCKING:
- * EH context.
- *
- * RETURNS:
- * 0 on success, -errno on failure.
- */
-void ata_acpi_associate(struct ata_host *host)
-{
-       int i, j;
-
-       if (!is_pci_dev(host->dev) || libata_noacpi)
-               return;
-
-       host->acpi_handle = DEVICE_ACPI_HANDLE(host->dev);
-       if (!host->acpi_handle)
-               return;
-
-       for (i = 0; i < host->n_ports; i++) {
-               struct ata_port *ap = host->ports[i];
-
-               if (host->ports[0]->flags & ATA_FLAG_ACPI_SATA)
-                       ata_acpi_associate_sata_port(ap);
-               else
-                       ata_acpi_associate_ide_port(ap);
-
-               if (ap->acpi_handle) {
-                       /* we might be on a docking station */
-                       register_hotplug_dock_device(ap->acpi_handle,
-                                                    &ata_acpi_ap_dock_ops, ap);
-               }
-
-               for (j = 0; j < ata_link_max_devices(&ap->link); j++) {
-                       struct ata_device *dev = &ap->link.device[j];
-
-                       if (dev->acpi_handle) {
-                               /* we might be on a docking station */
-                               register_hotplug_dock_device(dev->acpi_handle,
-                                                            &ata_acpi_dev_dock_ops, dev);
-                       }
-               }
-       }
-}
-
-/**
  * ata_acpi_dissociate - dissociate ATA host from ACPI objects
  * @host: target ATA host
  *
@@ -324,7 +232,7 @@ void ata_acpi_dissociate(struct ata_host
                struct ata_port *ap = host->ports[i];
                const struct ata_acpi_gtm *gtm = ata_acpi_init_gtm(ap);

-               if (ap->acpi_handle && gtm)
+               if (ata_ap_acpi_handle(ap) && gtm)
                        ata_acpi_stm(ap, gtm);
        }
 }
```

```
@@ -349,7 +257,8 @@ int ata_acpi_gtm(struct ata_port *ap, st
        acpi_status status;
        int rc = 0;

-       status = acpi_evaluate_object(ap->acpi_handle, "_GTM", NULL, &output);
+       status = acpi_evaluate_object(ata_ap_acpi_handle(ap), "_GTM", NULL,
+                                     &output);

        rc = -ENOENT;
        if (status == AE_NOT_FOUND)
@@ -419,7 +328,8 @@ int ata_acpi_stm(struct ata_port *ap, co
        input.count = 3;
        input.pointer = in_params;

-       status = acpi_evaluate_object(ap->acpi_handle, "_STM", &input, NULL);
+       status = acpi_evaluate_object(ata_ap_acpi_handle(ap), "_STM", &input,
+                                     NULL);

        if (status == AE_NOT_FOUND)
                return -ENOENT;
@@ -476,7 +386,8 @@ static int ata_dev_get_GTF(struct ata_de
                           __func__, ap->port_no);

        /* _GTF has no input parameters */
-       status = acpi_evaluate_object(dev->acpi_handle, "_GTF", NULL, &output);
+       status = acpi_evaluate_object(ata_dev_acpi_handle(dev), "_GTF", NULL,
+                                     &output);
        out_obj = dev->gtf_cache = output.pointer;

        if (ACPI_FAILURE(status)) {
@@ -842,7 +753,8 @@ static int ata_acpi_push_id(struct ata_d

        /* It's OK for _SDD to be missing too. */
        swap_buf_le16(dev->id, ATA_ID_WORDS);
-       status = acpi_evaluate_object(dev->acpi_handle, "_SDD", &input, NULL);
+       status = acpi_evaluate_object(ata_dev_acpi_handle(dev), "_SDD", &input,
+                                     NULL);
        swap_buf_le16(dev->id, ATA_ID_WORDS);

        if (status == AE_NOT_FOUND)
@@ -892,7 +804,7 @@ void ata_acpi_on_resume(struct ata_port
        const struct ata_acpi_gtm *gtm = ata_acpi_init_gtm(ap);
        struct ata_device *dev;

-       if (ap->acpi_handle && gtm) {
+       if (ata_ap_acpi_handle(ap) && gtm) {
                /* _GTM valid */

                /* restore timing parameters */
@@ -933,22 +845,22 @@ void ata_acpi_set_state(struct ata_port
 {
        struct ata_device *dev;

-       if (!ap->acpi_handle || (ap->flags & ATA_FLAG_ACPI_SATA))
+       if (!ata_ap_acpi_handle(ap) || (ap->flags & ATA_FLAG_ACPI_SATA))
                return;

        /* channel first and then drives for power on and vica versa
           for power off */
        if (state.event == PM_EVENT_ON)
-               acpi_bus_set_power(ap->acpi_handle, ACPI_STATE_D0);
+               acpi_bus_set_power(ata_ap_acpi_handle(ap), ACPI_STATE_D0);

        ata_for_each_dev(dev, &ap->link, ENABLED) {
-               if (dev->acpi_handle)
-                       acpi_bus_set_power(dev->acpi_handle,
+               if (ata_dev_acpi_handle(dev))
+                       acpi_bus_set_power(ata_dev_acpi_handle(dev),
```

```
                                        state.event == PM_EVENT_ON ?
                                                ACPI_STATE_D0 : ACPI_STATE_D3);
        }
        if (state.event != PM_EVENT_ON)
-               acpi_bus_set_power(ap->acpi_handle, ACPI_STATE_D3);
+               acpi_bus_set_power(ata_ap_acpi_handle(ap), ACPI_STATE_D3);
 }

 /**
@@ -973,7 +885,7 @@ int ata_acpi_on_devcfg(struct ata_device
        int nr_executed = 0;
        int rc;

-       if (!dev->acpi_handle)
+       if (!ata_dev_acpi_handle(dev))
                return 0;

        /* do we need to do _GTF? */
@@ -1019,7 +931,6 @@ int ata_acpi_on_devcfg(struct ata_device
        }

        ata_dev_warn(dev, "ACPI: failed the second time, disabled\n");
-       dev->acpi_handle = NULL;

        /* We can safely continue if no _GTF command has been executed
         * and port is not frozen.
@@ -1073,6 +984,9 @@ static int ata_acpi_bind_host(struct dev
        if (!*handle)
                return -ENODEV;

+       register_hotplug_dock_device(ata_ap_acpi_handle(ap),
+                                    &ata_acpi_ap_dock_ops, ap);
+
        return 0;
 }

@@ -1089,10 +1003,12 @@ static int ata_acpi_bind_device(struct d
        else
                ata_dev = &ap->link.device[id];

-       *handle = dev_acpi_handle(ata_dev);
+       *handle = ata_dev_acpi_handle(ata_dev);

        if (!*handle)
                return -ENODEV;
+       register_hotplug_dock_device(ata_dev_acpi_handle(ata_dev),
+                                    &ata_acpi_dev_dock_ops, ata_dev);

        return 0;
 }
Index: linux/drivers/ata/libata-core.c
===================================================================
--- linux.orig/drivers/ata/libata-core.c
+++ linux/drivers/ata/libata-core.c
@@ -5980,9 +5980,6 @@ int ata_host_register(struct ata_host *h
        if (rc)
                goto err_tadd;

-       /* associate with ACPI nodes */
-       ata_acpi_associate(host);
-
        /* set cable, sata_spd_limit and report */
        for (i = 0; i < host->n_ports; i++) {
                struct ata_port *ap = host->ports[i];
Index: linux/drivers/ata/libata-pmp.c
===================================================================
--- linux.orig/drivers/ata/libata-pmp.c
+++ linux/drivers/ata/libata-pmp.c
```

```
@@ -529,8 +529,6 @@ int sata_pmp_attach(struct ata_device *d
        ata_for_each_link(tlink, ap, EDGE)
                sata_link_init_spd(tlink);

-       ata_acpi_associate_sata_port(ap);
-
        return 0;

  fail:
@@ -570,8 +568,6 @@ static void sata_pmp_detach(struct ata_d
        ap->nr_pmp_links = 0;
        link->pmp = 0;
        spin_unlock_irqrestore(ap->lock, flags);
-
-       ata_acpi_associate_sata_port(ap);
 }

 /**
Index: linux/drivers/ata/libata.h
===================================================================
--- linux.orig/drivers/ata/libata.h
+++ linux/drivers/ata/libata.h
@@ -108,9 +108,6 @@ extern int ata_port_probe(struct ata_por
 /* libata-acpi.c */
 #ifdef CONFIG_ATA_ACPI
 extern unsigned int ata_acpi_gtf_filter;
-
-extern void ata_acpi_associate_sata_port(struct ata_port *ap);
-extern void ata_acpi_associate(struct ata_host *host);
 extern void ata_acpi_dissociate(struct ata_host *host);
 extern int ata_acpi_on_suspend(struct ata_port *ap);
 extern void ata_acpi_on_resume(struct ata_port *ap);
@@ -120,8 +117,6 @@ extern void ata_acpi_set_state(struct at
 extern int ata_acpi_register(void);
 extern void ata_acpi_unregister(void);
 #else
-static inline void ata_acpi_associate_sata_port(struct ata_port *ap) { }
-static inline void ata_acpi_associate(struct ata_host *host) { }
 static inline void ata_acpi_dissociate(struct ata_host *host) { }
 static inline int ata_acpi_on_suspend(struct ata_port *ap) { return 0; }
 static inline void ata_acpi_on_resume(struct ata_port *ap) { }
Index: linux/drivers/ata/pata_acpi.c
===================================================================
--- linux.orig/drivers/ata/pata_acpi.c
+++ linux/drivers/ata/pata_acpi.c
@@ -39,7 +39,7 @@ static int pacpi_pre_reset(struct ata_li
 {
        struct ata_port *ap = link->ap;
        struct pata_acpi *acpi = ap->private_data;
-       if (ap->acpi_handle == NULL || ata_acpi_gtm(ap, &acpi->gtm) < 0)
+       if (ata_ap_acpi_handle(ap) == NULL || ata_acpi_gtm(ap, &acpi->gtm) < 0)
                return -ENODEV;

        return ata_sff_prereset(link, deadline);
@@ -195,7 +195,7 @@ static int pacpi_port_start(struct ata_p
        struct pci_dev *pdev = to_pci_dev(ap->host->dev);
        struct pata_acpi *acpi;

-       if (ap->acpi_handle == NULL)
+       if (ata_ap_acpi_handle(ap) == NULL)
                return -ENODEV;

        acpi = ap->private_data = devm_kzalloc(&pdev->dev, sizeof(struct pata_acpi), GFP_KERNEL);
Index: linux/include/linux/libata.h
===================================================================
--- linux.orig/include/linux/libata.h
+++ linux/include/linux/libata.h
@@ -544,9 +544,6 @@ struct ata_host {
```

```
        struct mutex            eh_mutex;
        struct task_struct      *eh_owner;

-#ifdef CONFIG_ATA_ACPI
-       acpi_handle             acpi_handle;
-#endif
        struct ata_port         *simplex_claimed;       /* channel owning the DMA */
        struct ata_port         *ports[0];
 };
@@ -614,7 +611,6 @@ struct ata_device {
        struct scsi_device      *sdev;          /* attached SCSI device */
        void                    *private_data;
 #ifdef CONFIG_ATA_ACPI
-       acpi_handle             acpi_handle;
        union acpi_object       *gtf_cache;
        unsigned int            gtf_filter;
 #endif
@@ -796,7 +792,6 @@ struct ata_port {
        void                    *private_data;

 #ifdef CONFIG_ATA_ACPI
-       acpi_handle             acpi_handle;
        struct ata_acpi_gtm     __acpi_init_gtm; /* use ata_acpi_init_gtm() */
 #endif
        /* owned by EH */
@@ -1111,6 +1106,8 @@ int ata_acpi_stm(struct ata_port *ap, co
 int ata_acpi_gtm(struct ata_port *ap, struct ata_acpi_gtm *stm);
 unsigned long ata_acpi_gtm_xfermask(struct ata_device *dev,
                                    const struct ata_acpi_gtm *gtm);
+acpi_handle ata_ap_acpi_handle(struct ata_port *ap);
+acpi_handle ata_dev_acpi_handle(struct ata_device *dev);
 int ata_acpi_cbl_80wire(struct ata_port *ap, const struct ata_acpi_gtm *gtm);
 #else
 static inline const struct ata_acpi_gtm *ata_acpi_init_gtm(struct ata_port *ap)
```

[PATCH 4/7] acpi: Add support for linking docks to the objects they contain

```
Subject: [PATCH 4/7] acpi: Add support for linking docks to the objects they contain

From: Matthew Garrett <mjg@redhat.com>

When undocking, it's helpful to know which devices are going to
disappear. This patch adds support for adding symlinks to the device
into the docking bay.

Signed-off-by: Matthew Garrett <mjg@redhat.com>
Acked-by: Holger Macht <holger@homac.de>
---
 drivers/acpi/dock.c         |   78 ++++++++++++++++++++++++++++++++++++++++++++
 include/acpi/acpi_drivers.h |   10 +++++
 2 files changed, 88 insertions(+)

Index: linux/drivers/acpi/dock.c
===================================================================
--- linux.orig/drivers/acpi/dock.c
+++ linux/drivers/acpi/dock.c
@@ -276,6 +276,84 @@ int is_dock_device(acpi_handle handle)
 EXPORT_SYMBOL_GPL(is_dock_device);

 /**
+ * dock_link_device - link a device from the dock
+ * @handle: acpi handle of the potentially dependent device
+ */
+struct device **dock_link_device(acpi_handle handle)
```

```
+{
+       struct device *dev = acpi_get_physical_device(handle);
+       struct dock_station *dock_station;
+       int ret, dock = 0;
+       struct device **devices;
+
+       devices = kmalloc(dock_station_count * sizeof(struct device *),
+                         GFP_KERNEL);
+
+       if (!dev)
+               return NULL;
+
+       if (is_dock(handle)) {
+               put_device(dev);
+               return NULL;
+       }
+
+       list_for_each_entry(dock_station, &dock_stations, sibling) {
+               if (find_dock_dependent_device(dock_station, handle)) {
+                       ret = sysfs_create_link(&dock_station->dock_device->dev.kobj,
+                                               &dev->kobj, dev_name(dev));
+                       WARN_ON(ret);
+                       devices[dock] = &dock_station->dock_device->dev;
+                       dock++;
+               }
+       }
+       if (!dock)
+               put_device(dev);
+
+       devices[dock] = NULL;
+       return devices;
+}
+EXPORT_SYMBOL_GPL(dock_link_device);
+
+/**
+ * dock_unlink_device - unlink a device from the dock
+ * @handle: acpi handle of the potentially dependent device
+ */
+struct device **dock_unlink_device(acpi_handle handle)
+{
+       struct device *dev = acpi_get_physical_device(handle);
+       struct dock_station *dock_station;
+       int dock = 0;
+       struct device **devices =
+               kmalloc(dock_station_count * sizeof(struct device *),
+                       GFP_KERNEL);
+
+       if (!dev)
+               return NULL;
+
+       if (is_dock(handle)) {
+               put_device(dev);
+               return NULL;
+       }
+
+       list_for_each_entry(dock_station, &dock_stations, sibling) {
+               if (find_dock_dependent_device(dock_station, handle)) {
+                       sysfs_remove_link(&dock_station->dock_device->dev.kobj,
+                                         dev_name(dev));
+                       devices[dock] = &dock_station->dock_device->dev;
+                       dock++;
+               }
+       }
+       /* An extra reference has been held while the link existed */
+       if (dock)
+               put_device(dev);
+
+       put_device(dev);
```

```
+       devices[dock] = NULL;
+       return devices;
+}
+EXPORT_SYMBOL_GPL(dock_unlink_device);
+
+/**
  * dock_present - see if the dock station is present.
  * @ds: the dock station
  *
Index: linux/include/acpi/acpi_drivers.h
===================================================================
--- linux.orig/include/acpi/acpi_drivers.h
+++ linux/include/acpi/acpi_drivers.h
@@ -131,6 +131,8 @@ extern int register_hotplug_dock_device(
                                        const struct acpi_dock_ops *ops,
                                        void *context);
 extern void unregister_hotplug_dock_device(acpi_handle handle);
+extern struct device **dock_link_device(acpi_handle handle);
+extern struct device **dock_unlink_device(acpi_handle handle);
 #else
 static inline int is_dock_device(acpi_handle handle)
 {
@@ -152,6 +154,14 @@ static inline int register_hotplug_dock_
 static inline void unregister_hotplug_dock_device(acpi_handle handle)
 {
 }
+static inline struct device **dock_link_device(acpi_handle handle)
+{
+       return NULL;
+}
+static inline struct device **dock_unlink_device(acpi_handle handle)
+{
+       return NULL;
+}
 #endif

 #endif /*__ACPI_DRIVERS_H__*/
```

[PATCH 5/7] libata: Add links between removable devices and docks

```
Subject: [PATCH 5/7] libata: Add links between removable devices and docks

From: Matthew Garrett <mjg@redhat.com>

Attaching ata objects to docks makes it possible to identify which dock
should be used to trigger the removal of a device, and also allows
userspace to cleanly unmount filesystems before completing a
user-requested undocking.

Signed-off-by: Matthew Garrett <mjg@redhat.com>
Acked-by: Holger Macht <holger@homac.de>
---
 libata-acpi.c |   20 ++++++++++++++++++++
 libata-scsi.c |    3 +++
 libata.h      |    4 ++++
 3 files changed, 27 insertions(+)

Index: linux/drivers/ata/libata-acpi.c
===================================================================
--- linux.orig/drivers/ata/libata-acpi.c
+++ linux/drivers/ata/libata-acpi.c
@@ -955,6 +955,26 @@ void ata_acpi_on_disable(struct ata_devi
        ata_acpi_clear_gtf(dev);
 }
```

```
+void ata_acpi_bind_dock(struct ata_device *dev)
+{
+       struct device **docks;
+
+       if (!ata_dev_acpi_handle(dev))
+               return;
+       docks = dock_link_device(ata_dev_acpi_handle(dev));
+       kfree(docks);
+}
+
+void ata_acpi_unbind_dock(struct ata_device *dev)
+{
+       struct device **docks;
+
+       if (!ata_dev_acpi_handle(dev))
+               return;
+       docks = dock_unlink_device(ata_dev_acpi_handle(dev));
+       kfree(docks);
+}
+
 static int is_pci_ata(struct device *dev)
 {
        struct pci_dev *pdev;
Index: linux/drivers/ata/libata-scsi.c
===================================================================
--- linux.orig/drivers/ata/libata-scsi.c
+++ linux/drivers/ata/libata-scsi.c
@@ -3443,6 +3443,7 @@ void ata_scsi_scan_host(struct ata_port
                        if (!IS_ERR(sdev)) {
                                dev->sdev = sdev;
                                scsi_device_put(sdev);
+                               ata_acpi_bind_dock(dev);
                        } else {
                                dev->sdev = NULL;
                        }
@@ -3543,6 +3544,8 @@ static void ata_scsi_remove_dev(struct a
        sdev = dev->sdev;
        dev->sdev = NULL;

+       ata_acpi_unbind_dock(dev);
+
        if (sdev) {
                /* If user initiated unplug races with us, sdev can go
                 * away underneath us after the host lock and
Index: linux/drivers/ata/libata.h
===================================================================
--- linux.orig/drivers/ata/libata.h
+++ linux/drivers/ata/libata.h
@@ -116,6 +116,8 @@ extern void ata_acpi_on_disable(struct a
 extern void ata_acpi_set_state(struct ata_port *ap, pm_message_t state);
 extern int ata_acpi_register(void);
 extern void ata_acpi_unregister(void);
+extern void ata_acpi_bind_dock(struct ata_device *dev);
+extern void ata_acpi_unbind_dock(struct ata_device *dev);
 #else
 static inline void ata_acpi_dissociate(struct ata_host *host) { }
 static inline int ata_acpi_on_suspend(struct ata_port *ap) { return 0; }
@@ -126,6 +128,8 @@ static inline void ata_acpi_set_state(st
                                      pm_message_t state) { }
 static inline int ata_acpi_register(void) { return 0; }
 static void ata_acpi_unregister(void) { }
+static void ata_acpi_bind_dock(struct ata_device *dev) { }
+static void ata_acpi_unbind_dock(struct ata_device *dev) { }
 #endif

 /* libata-scsi.c */
```

[PATCH 6/7] libata: Generate and pass correct acpi handles

```
Subject: [PATCH 6/7] libata: Generate and pass correct acpi handles

Fix ACPI handle generation for device handles and pass the correct
handles to the dock driver.

Signed-off-by: Holger Macht <holger@homac.de>
---
 libata-acpi.c |   10 +++++-------
 1 file changed, 3 insertions(+), 7 deletions(-)

Index: linux/drivers/ata/libata-acpi.c
===================================================================
--- linux.orig/drivers/ata/libata-acpi.c
+++ linux/drivers/ata/libata-acpi.c
@@ -74,9 +74,6 @@ acpi_handle ata_dev_acpi_handle(struct a
        acpi_integer adr;
        struct ata_port *ap = dev->link->ap;

-       if (dev->sdev)
-               return DEVICE_ACPI_HANDLE(&dev->sdev->sdev_gendev);
-
        if (ap->flags & ATA_FLAG_ACPI_SATA) {
                if (!sata_pmp_attached(ap))
                        adr = SATA_ADR(ap->port_no, NO_PORT_MULT);
@@ -1004,8 +1001,7 @@ static int ata_acpi_bind_host(struct dev
        if (!*handle)
                return -ENODEV;

-       register_hotplug_dock_device(ata_ap_acpi_handle(ap),
-                                    &ata_acpi_ap_dock_ops, ap);
+       register_hotplug_dock_device(*handle, &ata_acpi_ap_dock_ops, ap);

        return 0;
 }
@@ -1027,8 +1023,8 @@ static int ata_acpi_bind_device(struct d

        if (!*handle)
                return -ENODEV;
-       register_hotplug_dock_device(ata_dev_acpi_handle(ata_dev),
-                                    &ata_acpi_dev_dock_ops, ata_dev);
+
+       register_hotplug_dock_device(*handle, &ata_acpi_dev_dock_ops, ata_dev);

        return 0;
 }
```

[PATCH 7/7] acpi: Prevent duplicate hotplug device registration on dock stations

```
Subject: [PATCH 7/7] acpi: Prevent duplicate hotplug device registration on dock stations

register_hotplug_dock_device() must only be called once per ACPI
handle. However, ACPI glue infrastructure does not allow to prevent
multiple find_device() invocations when a scsi bus or device appears or
disappears. So export a function from the dock driver to check if a
specific hotplug dock device is already registered and call this before
registering.

Signed-off-by: Holger Macht <holger@homac.de>
---
 drivers/acpi/dock.c          |   20 ++++++++++++++++++++
 drivers/ata/libata-acpi.c    |    6 ++++--
 include/acpi/acpi_drivers.h  |    1 +
```

```
 3 files changed, 25 insertions(+), 2 deletions(-)

Index: linux/drivers/acpi/dock.c
===================================================================
--- linux.orig/drivers/acpi/dock.c
+++ linux/drivers/acpi/dock.c
@@ -720,6 +720,26 @@ void unregister_hotplug_dock_device(acpi
 }
 EXPORT_SYMBOL_GPL(unregister_hotplug_dock_device);

+int is_registered_hotplug_dock_device(const struct acpi_dock_ops *ops)
+{
+       struct dock_dependent_device *dd;
+       struct dock_station *ds;
+
+       list_for_each_entry(ds, &dock_stations, sibling) {
+               mutex_lock(&ds->hp_lock);
+               list_for_each_entry(dd, &ds->hotplug_devices, hotplug_list) {
+                       if (ops == dd->ops) {
+                               mutex_unlock(&ds->hp_lock);
+                               return 1;
+                       }
+               }
+               mutex_unlock(&ds->hp_lock);
+       }
+
+       return 0;
+}
+EXPORT_SYMBOL(is_registered_hotplug_dock_device);
+
 /**
  * handle_eject_request - handle an undock request checking for error conditions
  *
Index: linux/drivers/ata/libata-acpi.c
===================================================================
--- linux.orig/drivers/ata/libata-acpi.c
+++ linux/drivers/ata/libata-acpi.c
@@ -1001,7 +1001,8 @@ static int ata_acpi_bind_host(struct dev
        if (!*handle)
                return -ENODEV;

-       register_hotplug_dock_device(*handle, &ata_acpi_ap_dock_ops, ap);
+       if (!is_registered_hotplug_dock_device(&ata_acpi_ap_dock_ops))
+               register_hotplug_dock_device(*handle, &ata_acpi_ap_dock_ops, ap);

        return 0;
 }
@@ -1024,7 +1025,8 @@ static int ata_acpi_bind_device(struct d
        if (!*handle)
                return -ENODEV;

-       register_hotplug_dock_device(*handle, &ata_acpi_dev_dock_ops, ata_dev);
+       if (!is_registered_hotplug_dock_device(&ata_acpi_dev_dock_ops))
+               register_hotplug_dock_device(*handle, &ata_acpi_dev_dock_ops, ata_dev);

        return 0;
 }
Index: linux/include/acpi/acpi_drivers.h
===================================================================
--- linux.orig/include/acpi/acpi_drivers.h
+++ linux/include/acpi/acpi_drivers.h
@@ -131,6 +131,7 @@ extern int register_hotplug_dock_device(
                                        const struct acpi_dock_ops *ops,
                                        void *context);
 extern void unregister_hotplug_dock_device(acpi_handle handle);
+extern int is_registered_hotplug_dock_device(const struct acpi_dock_ops *ops);
 extern struct device **dock_link_device(acpi_handle handle);
 extern struct device **dock_unlink_device(acpi_handle handle);
```

```
#else
```

A.1.2. Second Iteration of Patches (2012-01-20)

[PATCHv2 1/8] scsi: Add wrapper to access and set scsi_bus_type in struct acpi_bus_type

```
Subject: [PATCHv2 1/8] scsi: Add wrapper to access and set scsi_bus_type in struct acpi_bus_type

For being able to bind ata devices against acpi devices, scsi_bus_type
needs to be set as bus in struct acpi_bus_type. So add wrapper to
scsi_lib to accomplish that.

Signed-off-by: Holger Macht <holger@homac.de>
---
 drivers/scsi/scsi_lib.c |   11 ++++++++++
 include/scsi/scsi.h     |   10 ++++++++++
 2 files changed, 21 insertions(+), 0 deletions(-)

diff --git a/drivers/scsi/scsi_lib.c b/drivers/scsi/scsi_lib.c
index b2c95db..79f2654 100644
--- a/drivers/scsi/scsi_lib.c
+++ b/drivers/scsi/scsi_lib.c
@@ -68,6 +68,17 @@ static struct scsi_host_sg_pool scsi_sg_pools[] = {

 struct kmem_cache *scsi_sdb_cache;

+#ifdef CONFIG_ACPI
+#include <acpi/acpi_bus.h>
+
+int scsi_register_acpi_bus_type(struct acpi_bus_type *bus)
+{
+        bus->bus = &scsi_bus_type;
+        return register_acpi_bus_type(bus);
+}
+EXPORT_SYMBOL_GPL(scsi_register_acpi_bus_type);
+#endif
+
 /*
  * When to reinvoke queueing after a resource shortage. It's 3 msecs to
  * not change behaviour from the previous unplug mechanism, experimentation
diff --git a/include/scsi/scsi.h b/include/scsi/scsi.h
index 8001ae4..548fe9a 100644
--- a/include/scsi/scsi.h
+++ b/include/scsi/scsi.h
@@ -213,6 +213,16 @@ scsi_command_size(const unsigned char *cmnd)
                scsi_varlen_cdb_length(cmnd) : COMMAND_SIZE(cmnd[0]);
 }

+#ifdef CONFIG_ACPI
+struct acpi_bus_type;
+
+extern int
+scsi_register_acpi_bus_type(struct acpi_bus_type *bus);
+
+static inline void
+scsi_unregister_acpi_bus_type(struct acpi_bus_type *bus) { return; }
+#endif
+
 /*
  *  SCSI Architecture Model (SAM) Status codes. Taken from SAM-3 draft
  *  T10/1561-D Revision 4 Draft dated 7th November 2002.
-- 1.7.8
```

[PATCHv2 8/8] libata: Use correct PCI devices

```
Subject: [PATCHv2 8/8] libata: Use correct PCI devices

Commit 9a6d6a2ddabbd32c07f6a38b659e5f3db319fa5a made ata ports parent
devices of scsi hosts, so we need to go yet another level up to be able
to use the correct PCI devices.

Signed-off-by: Holger Macht <holger@homac.de>
---
 drivers/ata/libata-acpi.c |    6 +++---
 1 files changed, 3 insertions(+), 3 deletions(-)

diff --git a/drivers/ata/libata-acpi.c b/drivers/ata/libata-acpi.c
index a39f9b3..b03e468 100644
--- a/drivers/ata/libata-acpi.c
+++ b/drivers/ata/libata-acpi.c
@@ -996,7 +996,7 @@ static int ata_acpi_bind_host(struct device *dev, int host, acpi_handle *handle)
        if (ap->flags & ATA_FLAG_ACPI_SATA)
                return -ENODEV;

-       *handle = acpi_get_child(DEVICE_ACPI_HANDLE(dev->parent), ap->port_no);
+       *handle = acpi_get_child(DEVICE_ACPI_HANDLE(dev->parent->parent), ap->port_no);

        if (!*handle)
                return -ENODEV;
@@ -1036,13 +1036,13 @@ static int ata_acpi_find_device(struct device *dev, acpi_handle *handle)
        unsigned int host, channel, id, lun;

        if (sscanf(dev_name(dev), "host%u", &host) == 1) {
-               if (!is_pci_ata(dev->parent))
+               if (!is_pci_ata(dev->parent->parent))
                        return -ENODEV;

                return ata_acpi_bind_host(dev, host, handle);
        } else if (sscanf(dev_name(dev), "%d:%d:%d:%d",
                          &host, &channel, &id, &lun) == 4) {
-               if (!is_pci_ata(dev->parent->parent->parent))
+               if (!is_pci_ata(dev->parent->parent->parent->parent))
                        return -ENODEV;

                return ata_acpi_bind_device(dev, channel, id, handle);
-- 1.7.8
```

A.1.3. Additional Patches (2012-02-18)

[PATCH] dock: fix bootup oops and other dock_link breakage

```
[PATCH] dock: fix bootup oops and other dock_link breakage

From: Hugh Dickins <hughd <at> google.com>

dock_link_device() and dock_unlink_device() should bail out early
to avoid oops on zero-length kmalloc() when dock_station_count is 0.

But isn't there an off-by-one in that kmalloc() length anyway?
An extra NULL appended at the end suggests so.

Rework the ordering with gotos on failure to fix several issues.

And presumably dock_unlink_device() should be presenting the same
interface as dock_link_device(), with NULL returned when none found.

Signed-off-by: Hugh Dickins <hughd <at> google.com>
---
```

```
 drivers/acpi/dock.c |   69 ++++++++++++++++++++++++++++++++++++++-----------
 1 file changed, 49 insertions(+), 20 deletions(-)

--- linux-next/drivers/acpi/dock.c      2012-02-17 08:02:12.280064984 -0800
+++ fixed/drivers/acpi/dock.c   2012-02-18 09:57:54.926244796 -0800
@@ -281,21 +281,25 @@ EXPORT_SYMBOL_GPL(is_dock_device);
  */
 struct device **dock_link_device(acpi_handle handle)
 {
-       struct device *dev = acpi_get_physical_device(handle);
+       struct device *dev;
        struct dock_station *dock_station;
        int ret, dock = 0;
        struct device **devices;

-       devices = kmalloc(dock_station_count * sizeof(struct device *),
-                         GFP_KERNEL);
+       if (!dock_station_count)
+               return NULL;

-       if (!dev)
+       if (is_dock(handle))
                return NULL;

-       if (is_dock(handle)) {
-               put_device(dev);
+       dev = acpi_get_physical_device(handle);
+       if (!dev)
                return NULL;
-       }
+
+       devices = kmalloc((dock_station_count + 1) * sizeof(struct device *),
+                         GFP_KERNEL);
+       if (!devices)
+               goto put;

        list_for_each_entry(dock_station, &dock_stations, sibling) {
                if (find_dock_dependent_device(dock_station, handle)) {
@@ -304,13 +308,23 @@ struct device **dock_link_device(acpi_ha
                        WARN_ON(ret);
                        devices[dock] = &dock_station->dock_device->dev;
                        dock++;
+                       if (dock == dock_station_count)
+                               goto out;
                }
        }
-       if (!dock)
-               put_device(dev);

+       if (!dock)
+               goto free;
+out:
+       /* Keep a reference to the device while the link exists */
        devices[dock] = NULL;
        return devices;
+
+free:
+       kfree(devices);
+put:
+       put_device(dev);
+       return NULL;
 }
 EXPORT_SYMBOL_GPL(dock_link_device);

@@ -320,20 +334,25 @@ EXPORT_SYMBOL_GPL(dock_link_device);
  */
 struct device **dock_unlink_device(acpi_handle handle)
 {
```

```
-       struct device *dev = acpi_get_physical_device(handle);
+       struct device *dev;
        struct dock_station *dock_station;
        int dock = 0;
-       struct device **devices =
-               kmalloc(dock_station_count * sizeof(struct device *),
-                       GFP_KERNEL);
+       struct device **devices;

-       if (!dev)
+       if (!dock_station_count)
                return NULL;

-       if (is_dock(handle)) {
-               put_device(dev);
+       if (is_dock(handle))
                return NULL;
-       }
+
+       dev = acpi_get_physical_device(handle);
+       if (!dev)
+               return NULL;
+
+       devices = kmalloc((dock_station_count + 1) * sizeof(struct device *),
+                         GFP_KERNEL);
+       if (!devices)
+               goto put;

        list_for_each_entry(dock_station, &dock_stations, sibling) {
                if (find_dock_dependent_device(dock_station, handle)) {
@@ -341,15 +360,25 @@ struct device **dock_unlink_device(acpi_
                                                dev_name(dev));
                        devices[dock] = &dock_station->dock_device->dev;
                        dock++;
+                       if (dock == dock_station_count)
+                               goto out;
                }
        }
-       /* An extra reference has been held while the link existed */
-       if (dock)
-               put_device(dev);

+       if (!dock)
+               goto free;
+out:
+       /* An extra reference has been held while the link existed */
+       put_device(dev);
        put_device(dev);
        devices[dock] = NULL;
        return devices;
+
+free:
+       kfree(devices);
+put:
+       put_device(dev);
+       return NULL;
 }
 EXPORT_SYMBOL_GPL(dock_unlink_device);
```

[PATCH] acpi: Fix compiler error when setting CONFIG_ACPI_DOCK=n

```
Subject: [PATCH] acpi: Fix compiler error when setting CONFIG_ACPI_DOCK=n

When compiling with CONFIG_ACPI_DOCK=n,
is_registered_hotplug_dock_device() needs to be defined

Signed-off-by: Holger Macht <holger <at> homac.de>
```

```
---
 include/acpi/acpi_drivers.h |    4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)

diff --git a/include/acpi/acpi_drivers.h b/include/acpi/acpi_drivers.h
index 3c4e381..3319574 100644
--- a/include/acpi/acpi_drivers.h
+++ b/include/acpi/acpi_drivers.h
@@ -155,6 +155,10 @@ static inline int register_hotplug_dock_device(acpi_handle handle,
 static inline void unregister_hotplug_dock_device(acpi_handle handle)
 {
 }
+static inline int is_registered_hotplug_dock_device(const struct acpi_dock_ops *ops)
+{
+       return 0;
+}
 static inline struct device **dock_link_device(acpi_handle handle)
 {
        return NULL;
--
1.7.7
```

A.2. Patches for Userland

PATCH: Add new property DependingOnDockStation for drives

```
commit 2a7b8c14321ce38645ff44dff994e46e04a5a077
Author: Holger Macht <holger@homac.de>
Date:   Sun Jan 1 18:21:13 2012 +0100

    Add new property DependingOnDockStation for drives

    Add new property DependingOnDockStation which specifies whether a drive
    is depending on a dock station (e.g. on laptops). In other words,
    whether the drive is on the dock station, such as DVD drives. This way,
    desktops can safely unmount filesystems when they know they will
    disappear when an undock is requested.

    Signed-off-by: Holger Macht <holger@homac.de>

diff --git a/data/org.freedesktop.UDisks2.xml b/data/org.freedesktop.UDisks2.xml
index e88af62..f6da796 100644
--- a/data/org.freedesktop.UDisks2.xml
+++ b/data/org.freedesktop.UDisks2.xml
@@ -139,6 +139,11 @@
     -->
     <property name="MediaRemovable" type="b" access="read"/>

+    <!-- DependingOnDockStation:
+         Whether the drive is depending on a dock station (is on a dock station)
+    -->
+    <property name="DependingOnDockStation" type="b" access="read"/>
+
     <!-- MediaAvailable: Set to %FALSE if no medium is available.
          This is always %TRUE if #org.freedesktop.UDisks2.Drive:MediaChangeDetected is %FALSE.
     -->
diff --git a/src/udiskslinuxdrive.c b/src/udiskslinuxdrive.c
index cdfd281..5957bbb 100644
--- a/src/udiskslinuxdrive.c
+++ b/src/udiskslinuxdrive.c
@@ -195,6 +195,80 @@ ptr_str_array_compare (const gchar **a,
 }

 static void
+get_dock_dependent_devices(char         *dockdir,
+                           GSList       **list)
```

```
+{
+  GUdevDevice           *device;
+  GFileEnumerator       *file_enum;
+  GFileInfo             *info;
+  GFile                 *file;
+  GUdevClient           *client;
+
+  client = g_udev_client_new (NULL);
+  device = g_udev_client_query_by_sysfs_path (client, dockdir);
+
+  /* check if we have a dock_station */
+  const gchar * const *type = g_udev_device_get_sysfs_attr_as_strv (device, "type");
+  if (type == NULL)
+    return;
+  if (strcmp (type[0], "dock_station") != 0)
+    return;
+
+  file = g_file_new_for_path (dockdir);
+  file_enum = g_file_enumerate_children (file, G_FILE_ATTRIBUTE_STANDARD_NAME ","
+                                         G_FILE_ATTRIBUTE_STANDARD_IS_SYMLINK ","
+                                         G_FILE_ATTRIBUTE_STANDARD_SYMLINK_TARGET,
+                                         0, NULL, 0);
+  if (file_enum == NULL)
+    return;
+
+  while ((info = g_file_enumerator_next_file (file_enum, NULL, 0)) != NULL) {
+    const gchar *target = g_file_info_get_symlink_target (info);
+    gchar *pwd;
+
+    if (target == NULL)
+      continue;
+
+    if (!g_regex_match_simple ("[0-9]:[0-9]:[0-9:[0-9]", g_file_info_get_name (info), 0, 0))
+      continue;
+
+    pwd = g_get_current_dir (); /* save current directory */
+    g_chdir (dockdir); /* needed for realpath() to work */
+
+    *list = g_slist_append (*list, realpath (target, 0));
+
+    g_chdir (pwd); /* restore previous directory */
+    g_free (pwd);
+  }
+}
+
+static gboolean
+device_is_depending_on_dock_station(GUdevDevice *device)
+{
+  GSList *it, *list = NULL;
+  gchar *dockdir;
+  int i = 0;
+
+  while (TRUE) {
+    dockdir = g_strdup_printf ("/sys/devices/platform/dock.%u", i);
+    int exists = g_file_test (dockdir, G_FILE_TEST_EXISTS) && g_file_test (dockdir, G_FILE_TEST_IS_DI
+    g_free (dockdir);
+
+    if (exists)
+      get_dock_dependent_devices (dockdir, &list);
+    else
+      break;
+    i++;
+  }
+
+  for (it = list; it != NULL; it = g_slist_next(it))
+    if (g_str_has_prefix (g_udev_device_get_sysfs_path (device), it->data))
+      return TRUE;
+
```

```
+  return FALSE;
+}
+
+static void
 set_media (UDisksDrive       *iface,
            GUdevDevice       *device)
 {
@@ -229,6 +303,7 @@ set_media (UDisksDrive       *iface,
     removable = FALSE;
   udisks_drive_set_media_removable (iface, removable);
   udisks_drive_set_ejectable (iface, kernel_removable);
+  udisks_drive_set_depending_on_dock_station (iface, device_is_depending_on_dock_station (device));

   media_in_drive = NULL;
   if (udisks_drive_get_media_available (iface))
```

Bibliography

[1] *Advanced Configuration and Power Interface Specification*, October 2006. http://www.acpi.info.

[2] The Linux Kernel Community. File server for recent and archived kernel releases, 2012. [Online; accessed 21-May-2012] `http://www.kernel.org/pub/linux/kernel/`.

[3] The Linux Kernel Community. Kernel Documentation/ManagementStyle, 2012. [Online; accessed 14-May-2012].

[4] The Linux Kernel Community. Kernel Documentation/SubmittingPatches, 2012. [Online; accessed 14-May-2012].

[5] Jonathan Corbet. How to Participate in the Linux Community, 2008. [Online; accessed 23-May-2012].

[6] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting Empirical Methods for Software Engineering Research Guide to Advanced Empirical Software Engineering. In Forrest Shull and Janice Singer, editors, *Guide to Advanced Empirical Software Engineering*, chapter 11, pages 285–311. Springer London, London, 2008.

[7] David G. Glance. Release criteria for the Linux kernel. *First Monday*, 9, number 4, 2004.

[8] Andi Kleen. On submitting kernel patches. In *Proceedings of the Linux Symposium*, Ottawa, Ontario Canada, 23-26 July 2008.

[9] Gwendolyn K. Lee and Robert E. Cole. From a firm-based to a community-based model of knowledge creation: The case of the linux kernel development. *Organization Science*, 14(6):633–649, November 2003.

[10] R. Love. *Linux kernel development*. Novell Press Series. Novell Press, 2005.

[11] A Mockus, R T Fielding, and J Herbsleb. *A case study of open source software development: the Apache server*, volume 0, pages 263–272. Acm, 2000.

[12] Andrew Morton. The perfect patch, 2012. [Online; accessed 20-May-2012].

[13] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[14] Peter Weißgerber, Daniel Neu, and Stephan Diehl. Small patches get in! In *Proceedings of the 2008 international working conference on Mining software repositories*, MSR '08, pages 67–76, New York, NY, USA, 2008. ACM.

[15] Wikipedia. User space — Wikipedia, The Free Encyclopedia, 2011. [Online; accessed 25-May-2012] http://en.wikipedia.org/wiki/User_space.

[16] Wikipedia. Advanced Configuration and Power Interface — Wikipedia, The Free Encyclopedia, 2012. [Online; accessed 7-May-2012] `http://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface`.

[17] Wikipedia. GNOME — Wikipedia, The Free Encyclopedia, 2012. [Online; accessed 15-May-2012] http://en.wikipedia.org/wiki/GNOME.

[18] Wikipedia. Revision control — Wikipedia, The Free Encyclopedia, 2012. [Online; accessed 25-May-2012] http://en.wikipedia.org/wiki/Revision_control.