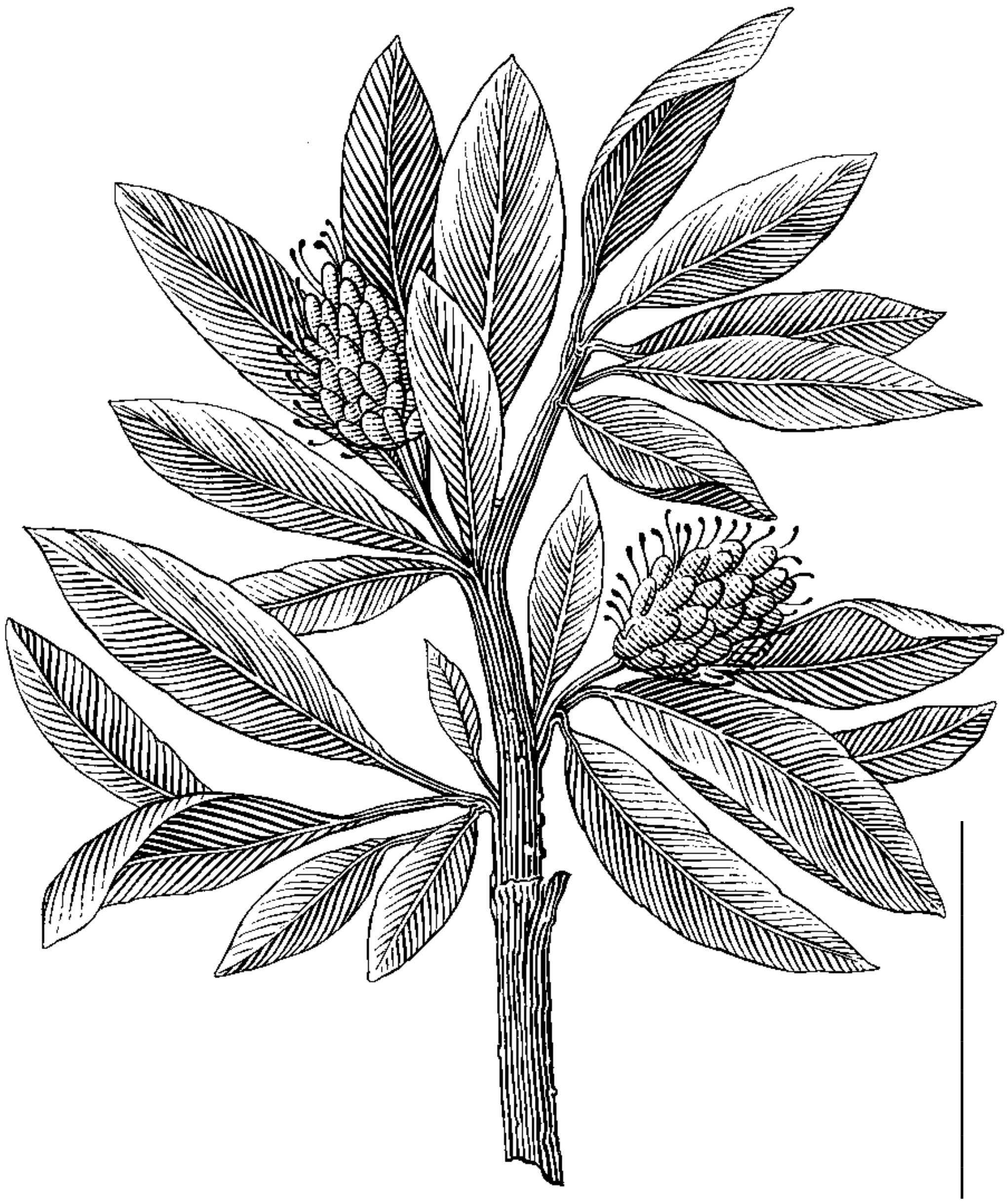
Linnéuniversitetet
Institutionen för datavetenskap, fysik och matematik

Master project

# Universal Tagging



Author: Joakim Söderlund
Semester: VT13
Subject: Computer Science
Level: Master
Course code: 5DV00E

## Abstract

The goal of this master degree project was to find out whether or not a highly integrated system for universal tagging of content improves the usability of a desktop environment. A prototype of such a system was implemented and integrated into the GNOME desktop environment. A usability study was then performed which showed that the tagging system did improve the usability of the desktop environment.

## Keywords

universal tagging, tag, usability, desktop environment, gnome, free software

## Acknowledgements

- **Ola Petersson** - For supervising the project.
- **Jessica Allan** - For proofreading the report.
- **The Testers** - For participating in the usability study.
- **The GNOME Developers** - For an excellent desktop environment.

# Contents

# 1 Introduction

Locating content previously viewed or created in a computer can be a tedious task. It involves either knowing exactly where something is located or using various search utilities. Additionally, different kinds of content must often be located in its own very specific way. For example, searching for local files is done via a file manager, while searching for web pages is done via an online search engine.

Bringing up different applications and services depending on the type of content can be both time consuming and inconvenient. An easily accessible system for locating all kinds of content could potentially improve the usability of a desktop environment.

This master degree project will focus on tagging as a means to organize, search, and browse any type of content within the GNOME desktop environment.

## 1.1 Goal

The goal of this master degree project is to find out whether or not a highly integrated system for universal tagging of content improves the usability of a desktop environment.

## 1.2 Method

To help answer the question, a highly integrated system for universal tagging of content, called TagPaca, will be implemented for the GNOME desktop environment. TagPaca will then be used in a usability study to determine whether or not it has improved the usability of the desktop environment.

## 1.3 Scope

This project primarily focuses on finding content using a tagging system. While the system will support manual tagging, automatic tagging has been left out due to time constraints. Additionally, manual tagging will be left out of the usability study as it is difficult to prove whether or not a user has tagged content correctly. A user may also prefer to organize his or her content in a way that differs from other users, thus turning tags into something personal rather than something definite.

## 1.4 Structure

This report will first describe the GNOME desktop environment and some of the general ideas behind the tagging system. The report will then cover some already existing technologies which also attempts to solve the task of making previously viewed or created content easier to locate.

Later on, each individual part of the tagging system's user interface is shown. The report then describes the design and implementation of TagPaca. After that, the usability study will be presented and its results interpreted. Finally, a conclusion will be provided along with suggestions for future work.

# 2 Project Plan

This section describes why this project targets the GNOME desktop environment and why it focuses on tags.

## 2.1 GNOME

GNOME is a desktop environment which provides fundamental tools for using computers[13]. The desktop environment includes a desktop shell (figure 2.1), a file manager, a browser, an e-mail client, a text editor, a photo manager, and more. GNOME is free software and runs on several UNIX-like operating systems such as Linux, BSD, and Solaris.

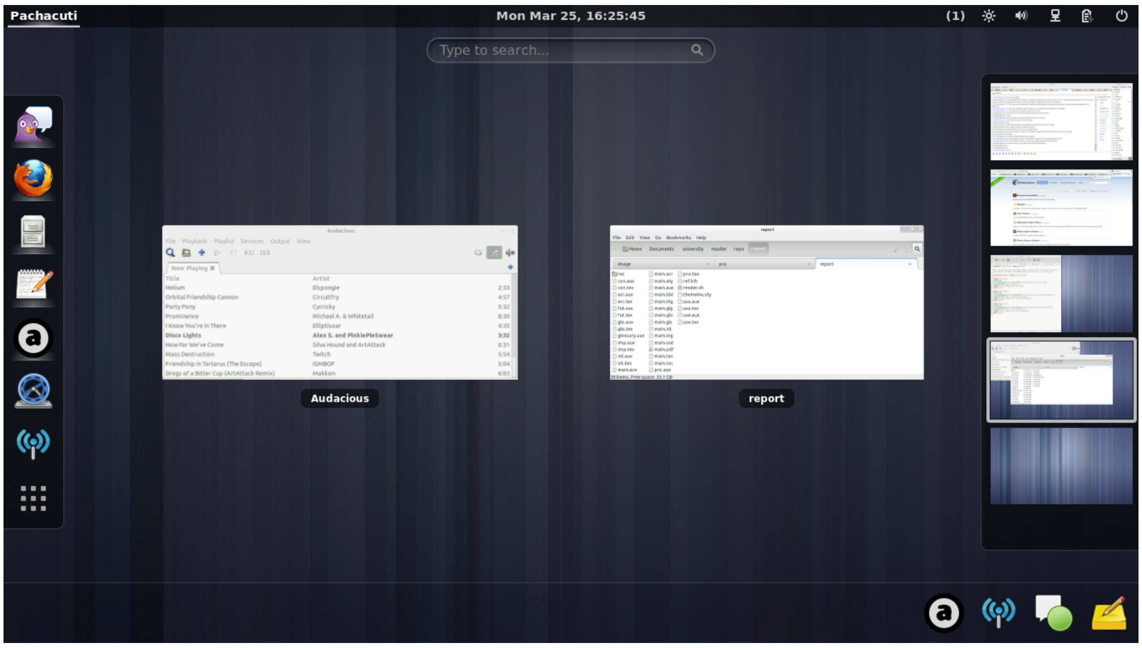


Figure 2.1: The GNOME Shell overview.

Many of the GNOME projects, including GNOME Shell[14], have support for extensions. This means that programs can be modified on the fly without having to change their source code. As an example, consider the GNOME desktop in figure 2.2 and compare it to figure 2.3. The first screenshot shows the desktop as it is without extensions, while the second shows it with a number of extensions enabled. With extensions enabled, the top panel has become transparent and its rounded corners have been removed. The contents of the panel has also been changed. Additionally, the desktop has gained two new panels, namely a dock and a window list. Although extensive, these are all changes that do not require modifying the GNOME Shell source code.

This brings us to the reasons why GNOME was chosen as the target platform for this degree project. The extension support allows for integrating much of TagPaca without modifying the desktop environment's source code. Still, writing extensions may not always be enough. Because GNOME is free software it is also possible to modify its source code to fully make it meet the goals of this degree project. Another reason why GNOME was chosen is because it is the author's desktop environment of choice for day-to-day computer use.

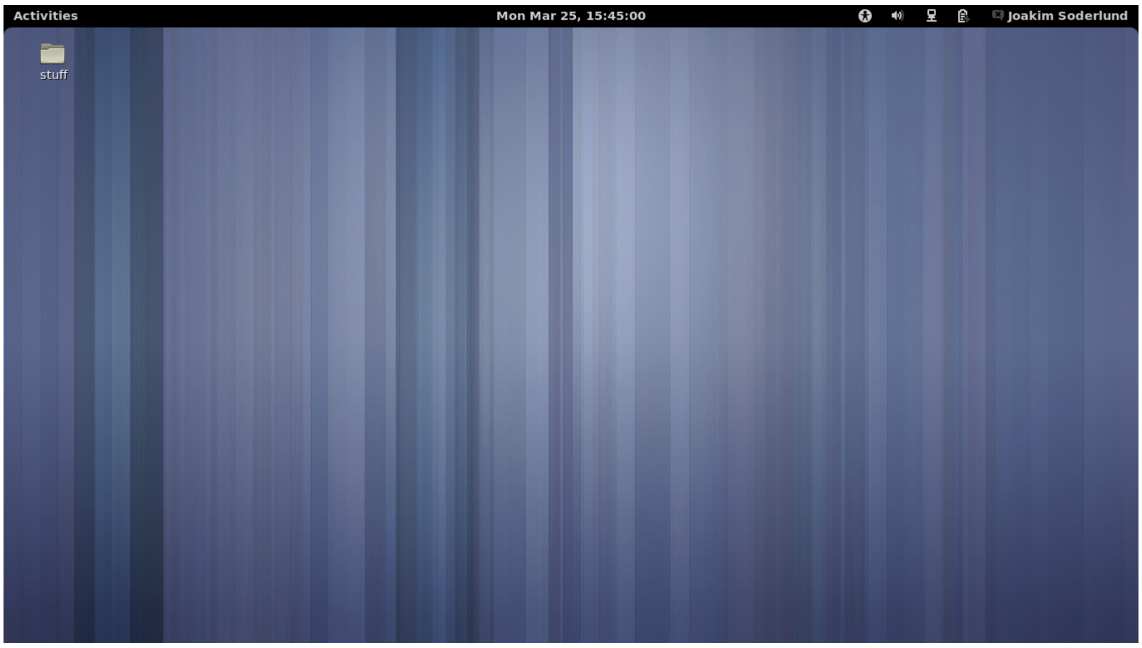


Figure 2.2: The GNOME Shell desktop without extensions.

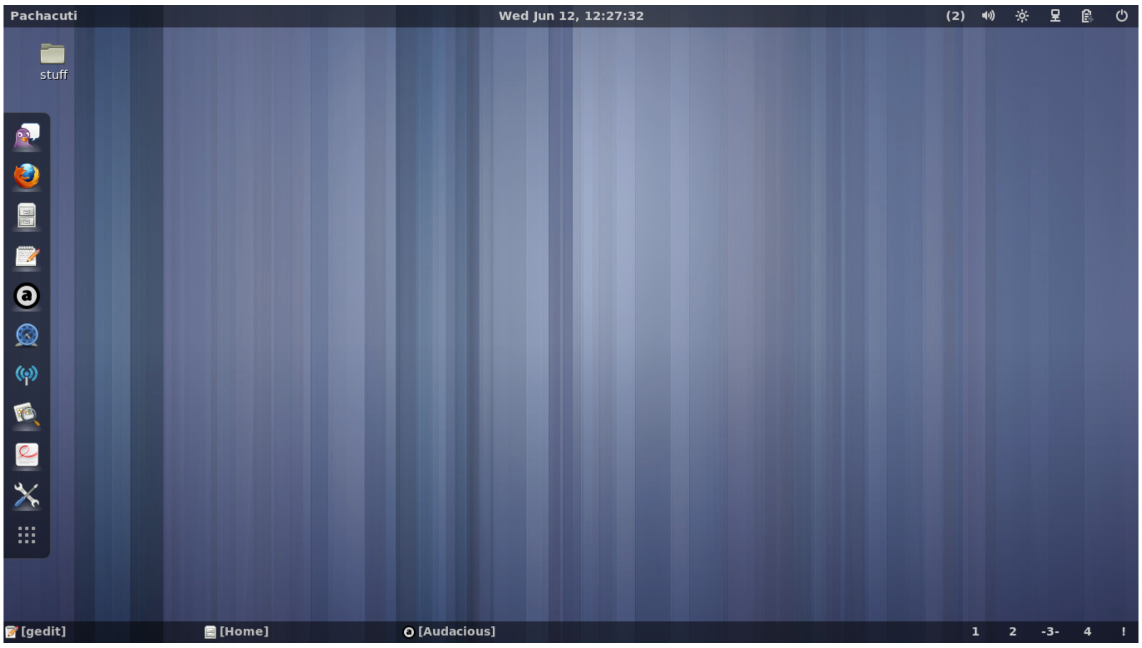


Figure 2.3: The GNOME Shell desktop with extensions.

A proprietary desktop environment such as that in Windows or OS X could have been chosen instead of GNOME. However, not having access to the source code could have made TagPaca difficult to integrate. For example, had it not been possible to modify GNOME, then tagging via drag and drop (section 4.3) would have been far more difficult to implement (section 5.8.1). Other free software desktop environments such as KDE and Xfce could also have been used. GNOME was however still chosen over both the free and the proprietary alternatives on the grounds of familiarity.

## 2.2 Tags

Tagging gives users a way to attach labels to their content, making it easier to find later on. In a way, tagging is similar to directories in file systems as it gives users the ability to control how their content is organized. The main difference is that users are not limited to a hierarchical structure.

As an example, consider a desktop wallpaper of an alpaca standing on a beach in front of an endless ocean and a setting sun. With tags, the user could simply attach all these aspects to the image, tagging it with “wallpaper”, “alpaca”, “beach”, “ocean”, and “sunset”. This could become a bit problematic with directories. Imagine if the user already has the directories “alpaca”, “wallpaper”, and “sunset”. To which directory does the image belong? Should each directory have a copy or link to the image? What about subdirectories for some finer granularity? While users still find answers to these questions, tags can help by eliminating the questions altogether.

Another problem arises when the user wants to find the image again. Imagine if the user wants to view all pictures of alpacas. With tags, the user would be presented with the previously mentioned image when performing a search for the “alpaca” tag. The user may however have chosen to store the image in the “wallpaper” directory. Unless a subdirectory or the image’s file name contains the word “alpaca”, the image is not likely to be part of a path-based search result.

Indexers could also help users find content. Indexing content such as arbitrary images is however not a trivial task. Additionally, a search for e-mails about Italy may not return messages about hotels in Italy if the country was not explicitly mentioned. While a more sophisticated indexer may be able to solve such problems, combining an indexer with a tagging system could prove useful. The indexer could help the tagging system by automatically applying tags to content. Problems such as the search for e-mails could be solved by introducing relationships between the tags. For example, hotel tags could have a “located in” relationship with a city tag, which in turn could have a “located in” relationship with a country tag.

Although interesting, automatic tagging is outside the scope of this degree project. As with the hierarchical directory structure of file systems, complete control over the organization of content is given to the user. The improvement the tagging system provides for files is therefore to free them from the limitations of a hierarchical structure.

## 2.3 Universal

Another goal with the tagging system is being able to handle any content type. Instead of users having to use multiple programs to find different kinds of content, TagPaca should attempt to support all of them. The prototype will at the very least support files, e-mails and web pages.

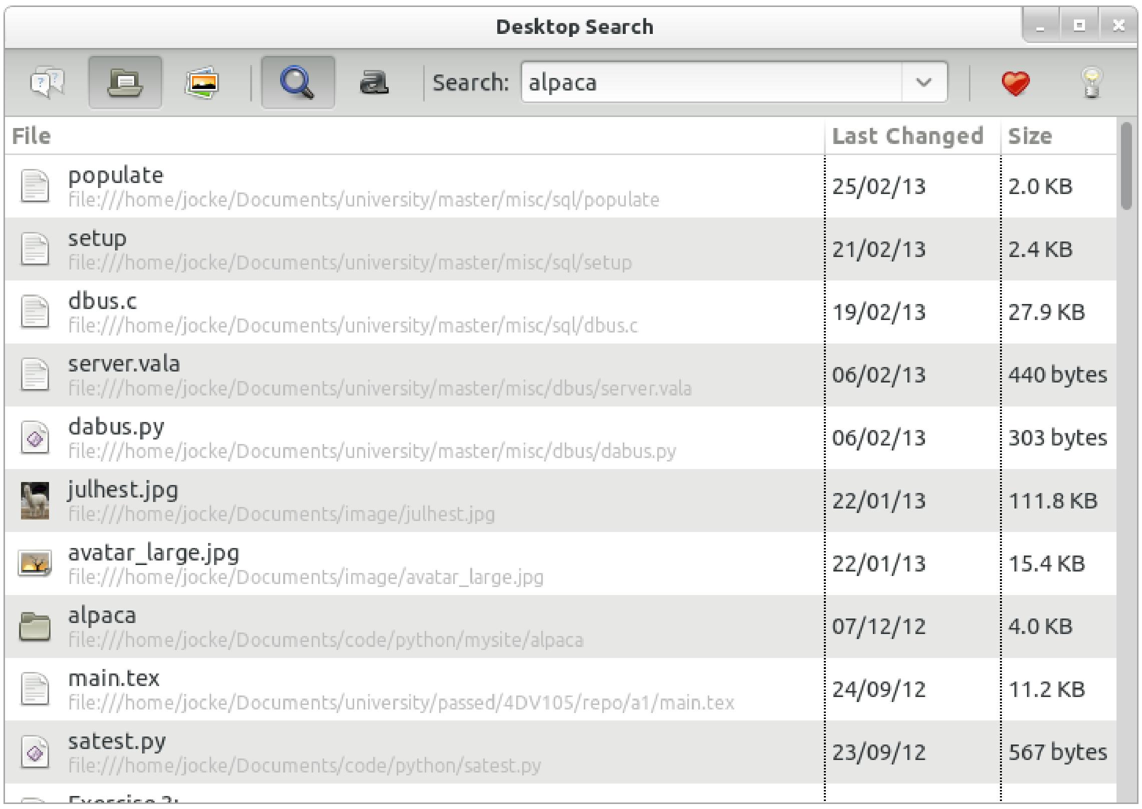
This goal introduces quite a few implementation problems. Examples include remote items becoming unavailable, users switching e-mail client, and different item types having different meta data. However, being able to find multiple content types using the same system can save users both time and effort (section 6.4.3). The alternative would have been to focus on a single content type. This would have made the implementation far easier, but also made the system less useful to its users. The additional problems seem worth the effort as making content easier to find is a key aspect of the tagging system.

# 3 Existing Technologies

This section describes some existing technologies for GNOME with goals similar to this project. Differences between this project and existing technologies are also discussed.

## 3.1 Tracker

Tracker is an indexer[6] which gathers information about content. The software runs in the background indexing files, e-mails, and other data. Tracker provides libraries, D-Bus interfaces, and applications (figure 3.1) for querying the collected data.



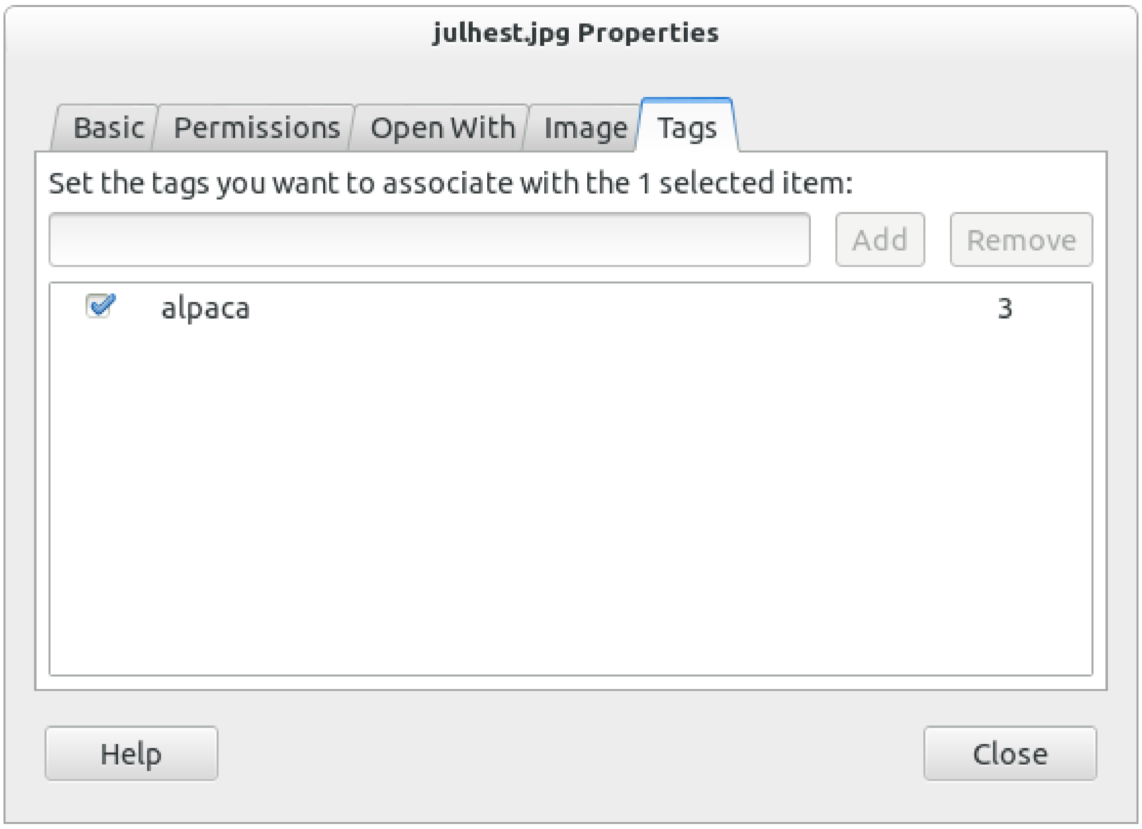
| File | Last Changed | Size |
|---|---|---|
| populate<br>file:///home/jocke/Documents/university/master/misc/sql/populate | 25/02/13 | 2.0 KB |
| setup<br>file:///home/jocke/Documents/university/master/misc/sql/setup | 21/02/13 | 2.4 KB |
| dbus.c<br>file:///home/jocke/Documents/university/master/misc/sql/dbus.c | 19/02/13 | 27.9 KB |
| server.vala<br>file:///home/jocke/Documents/university/master/misc/dbus/server.vala | 06/02/13 | 440 bytes |
| dabus.py<br>file:///home/jocke/Documents/university/master/misc/dbus/dabus.py | 06/02/13 | 303 bytes |
| julhest.jpg<br>file:///home/jocke/Documents/image/julhest.jpg | 22/01/13 | 111.8 KB |
| avatar_large.jpg<br>file:///home/jocke/Documents/image/avatar_large.jpg | 22/01/13 | 15.4 KB |
| alpaca<br>file:///home/jocke/Documents/code/python/mysite/alpaca | 07/12/12 | 4.0 KB |
| main.tex<br>file:///home/jocke/Documents/university/passed/4DV105/repo/a1/main.tex | 24/09/12 | 11.2 KB |
| satest.py<br>file:///home/jocke/Documents/code/python/satest.py | 23/09/12 | 567 bytes |

Figure 3.1: Tracker Desktop Search.

Multiple applications can use the collected data as a centralized index. For example, media players could query Tracker's database instead of each maintaining their own indexes. Although focused on indexing, Tracker also has support for tagging (figure 3.2).

At the time of writing, end users cannot easily utilize this data directly to find content. To search for content the user either has to start an external program called Desktop Search (figure 3.1) or use a GNOME Shell search extension (figure 3.3). Neither of those user interfaces provide an obvious way to browse the content other than searching by text.

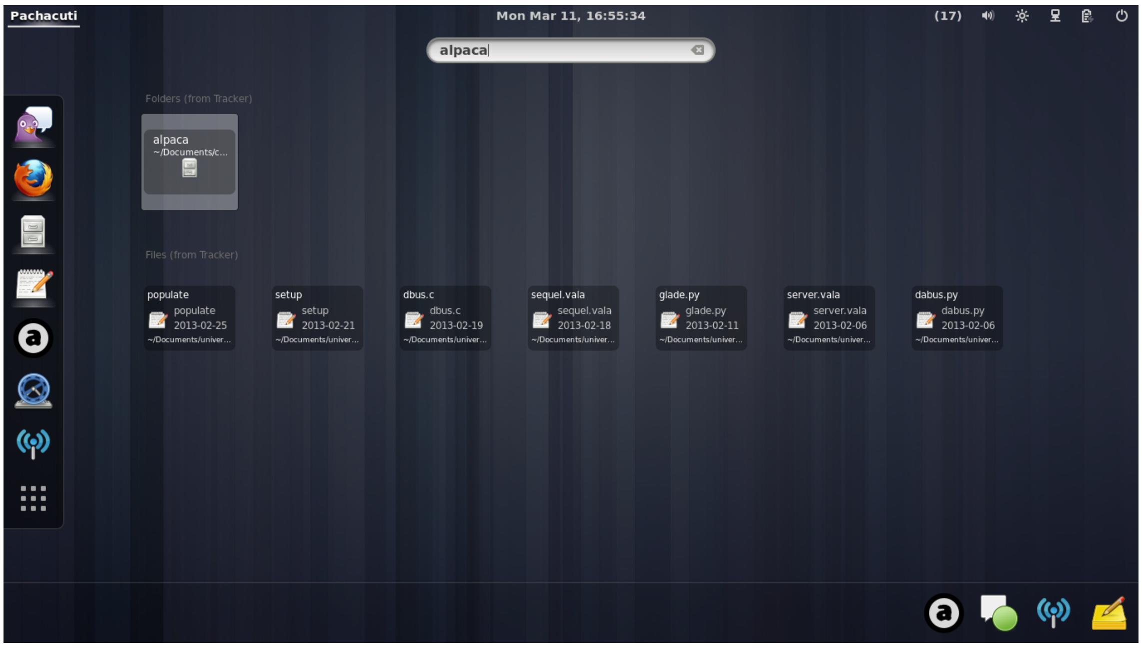
Tracker was under consideration to be extended for use in this degree project. This may have been beneficial both to Tracker (patch submissions) and this degree project (code reuse). However, due to the fact that the main focus of Tracker is indexing, modifying it to instead focus on tagging may have been difficult. Tracker is also a fairly large system[25], so modifying it may have required too much effort to be feasible. Mimicking the way Tracker stores data and queries its database was also considered. However, the idea was rejected due to the author's lack of experience with both SPARQL and NEPOMUK which are used by Tracker.



Set the tags you want to associate with the 1 selected item:

| | | |
|---|---|---|
| ✓ | alpaca | 3 |

Figure 3.2: Tracker Nautilus extension for tagging files.



Folders (from Tracker)

alpaca
~/Documents/c...

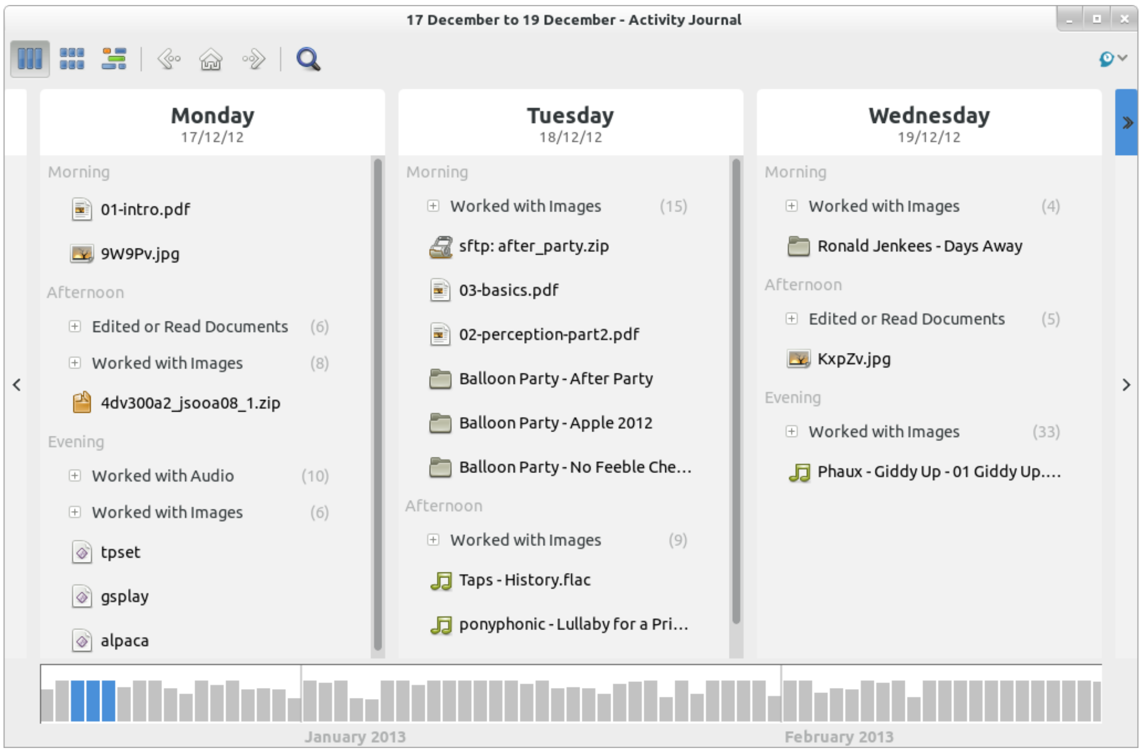
Files (from Tracker)

| | | | | | | |
|---|---|---|---|---|---|---|
| populate | setup | dbus.c | sequel.vala | glade.py | server.vala | dabus.py |
| populate | setup | dbus.c | sequel.vala | glade.py | server.vala | dabus.py |
| 2013-02-25 | 2013-02-21 | 2013-02-19 | 2013-02-18 | 2013-02-11 | 2013-02-06 | 2013-02-06 |
| ~/Documents/univer... | ~/Documents/univer... | ~/Documents/univer... | ~/Documents/univer... | ~/Documents/univer... | ~/Documents/univer... | ~/Documents/univer... |

Figure 3.3: Tracker GNOME Shell integration.

## 3.2 Zeitgeist

Zeitgeist can be used to log user activity such as opened files, conversations, and visited websites[28]. Zeitgeist also attempts to find relationships between content depending on how or when it was used. GNOME Activity Journal (figure 3.4) can be used to view the data collected by Zeitgeist.


17 December to 19 December - Activity Journal

**Monday** 17/12/12

Morning
- 01-intro.pdf
- 9W9Pv.jpg

Afternoon
- Edited or Read Documents (6)
- Worked with Images (8)
- 4dv300a2_jsooa08_1.zip

Evening
- Worked with Audio (10)
- Worked with Images (6)
- tpset
- gsplay
- alpaca

**Tuesday** 18/12/12

Morning
- Worked with Images (15)
- sftp: after_party.zip
- 03-basics.pdf
- 02-perception-part2.pdf
- Balloon Party - After Party
- Balloon Party - Apple 2012
- Balloon Party - No Feeble Che...

Afternoon
- Worked with Images (9)
- Taps - History.flac
- ponyphonic - Lullaby for a Pri...

**Wednesday** 19/12/12

Morning
- Worked with Images (4)
- Ronald Jenkees - Days Away

Afternoon
- Edited or Read Documents (5)
- KxpZv.jpg

Evening
- Worked with Images (33)
- Phaux - Giddy Up - 01 Giddy Up....

Figure 3.4: GNOME Activity Journal.

There are also several GNOME Shell extensions for Zeitgeist. Jump Lists (figure 3.5) allows users to open the most recently used content based on which application it was used in. Journal (figure 3.6) also shows the most recently used content, but bases its results on content types rather than applications.

## 3.3 Alternatives

There are several alternatives to the previously mentioned existing technologies. Some examples being image tagging in Windows 7[30], file tagging in OS X[31], and the Nepomuk support in KDE[29]. These systems are not discussed in further detail because they are not targeting the GNOME desktop environment. The built in tagging solutions in Windows and OS X are also not open source, so taking a closer look at their implementations would not be possible.

## 3.4 TagPaca

TagPaca will draw upon features from both Tracker and Zeitgeist. Tagging, like in Tracker, will be used to help users organize and find content. A timeline, similar to that in GNOME Activity Journal, will be used to help users find content depending on when it was added or modified. Automatic tagging and indexing would have been useful, but will not be implemented due to time constraints.

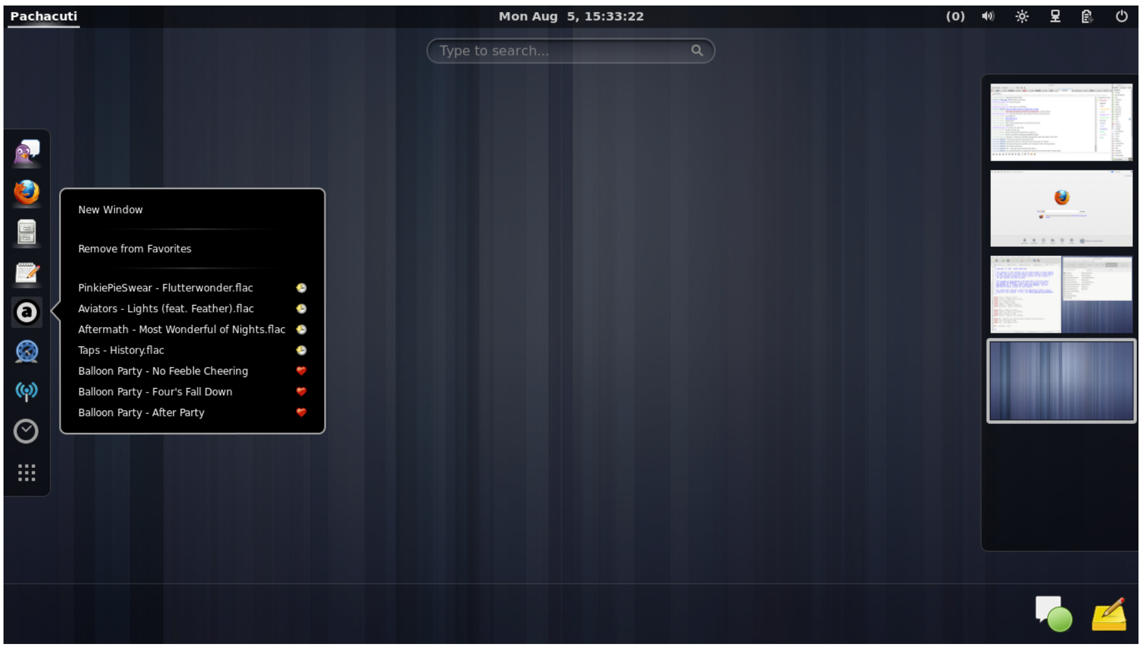

Pachacuti
Mon Aug 5, 15:33:22
Type to search...
New Window
Remove from Favorites
PinkiePieSwear - Flutterwonder.flac
Aviators - Lights (feat. Feather).flac
Aftermath - Most Wonderful of Nights.flac
Taps - History.flac
Balloon Party - No Feeble Cheering
Balloon Party - Four's Fall Down
Balloon Party - After Party

Figure 3.5: Zeitgeist GNOME Shell Jump Lists.

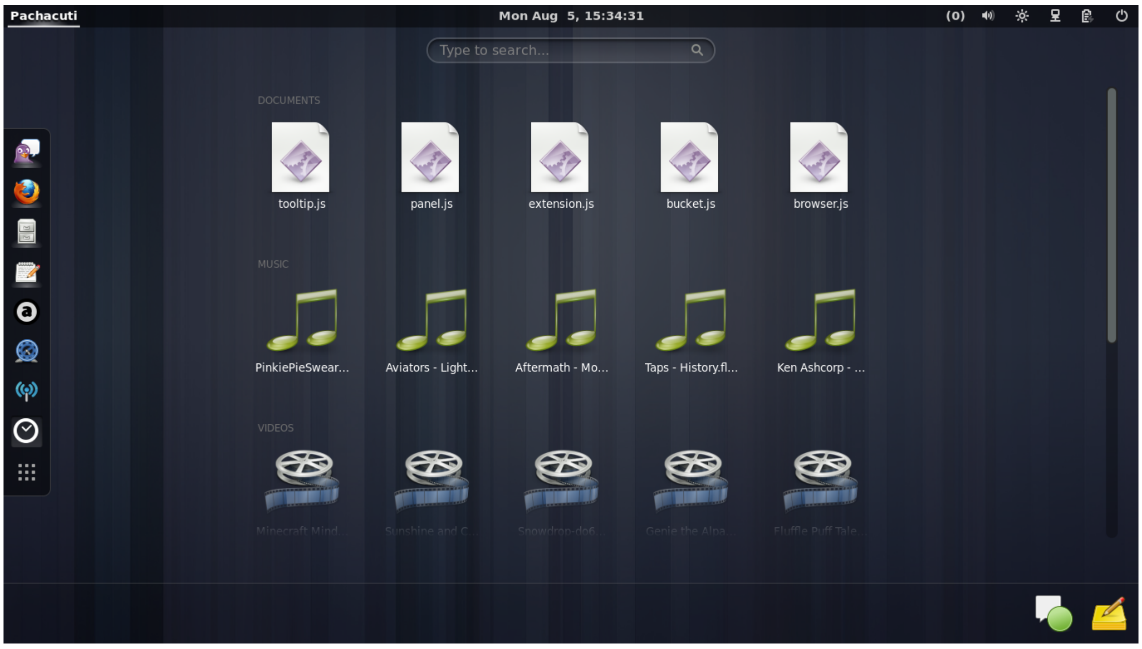


Figure 3.6: Zeitgeist GNOME Shell Journal.

# 4 User Interface

This section describes the design of the user interface and how it is used by users.

## 4.1 Features

TagPaca was developed using an experimental approach. At first there was only a general idea of which features were to be included. The system then slowly evolved into its final form as time progressed. The user interface now supports tagging of content, browsing content by tag, browsing content by date, persistent browsing states, and simple plain text notepads.

## 4.2 Extension

The user interface of TagPaca was created as an extension for GNOME Shell[15]. The extension was created in JavaScript via Gjs[12] using St[22], Clutter[4], and GNOME Shell widgets. Animations were created using the Tweener API[26] provided by GNOME Shell. See section 5 for more information about the implementation.

## 4.3 Panel

The user interface was implemented as an additional panel (figure 4.1) for GNOME Shell. This panel is always directly accessible regardless of what the user is doing at any given time. To access the panel, the user simply has to move his or her mouse cursor to the right edge of the main monitor. Doing so will cause the TagPaca panel to become visible.

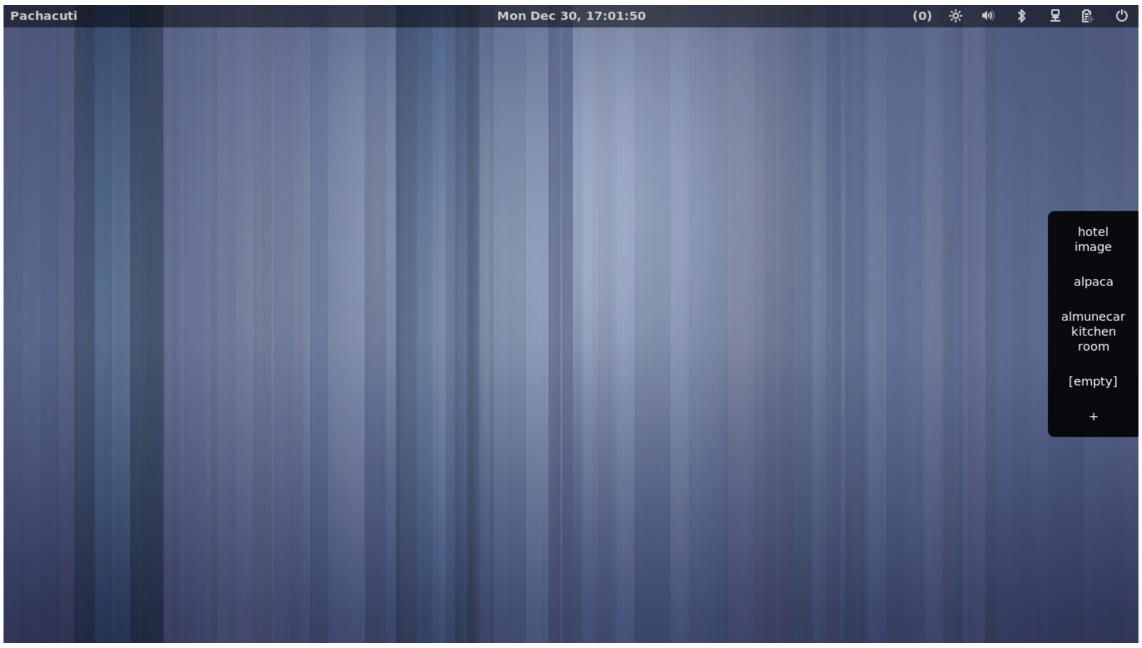
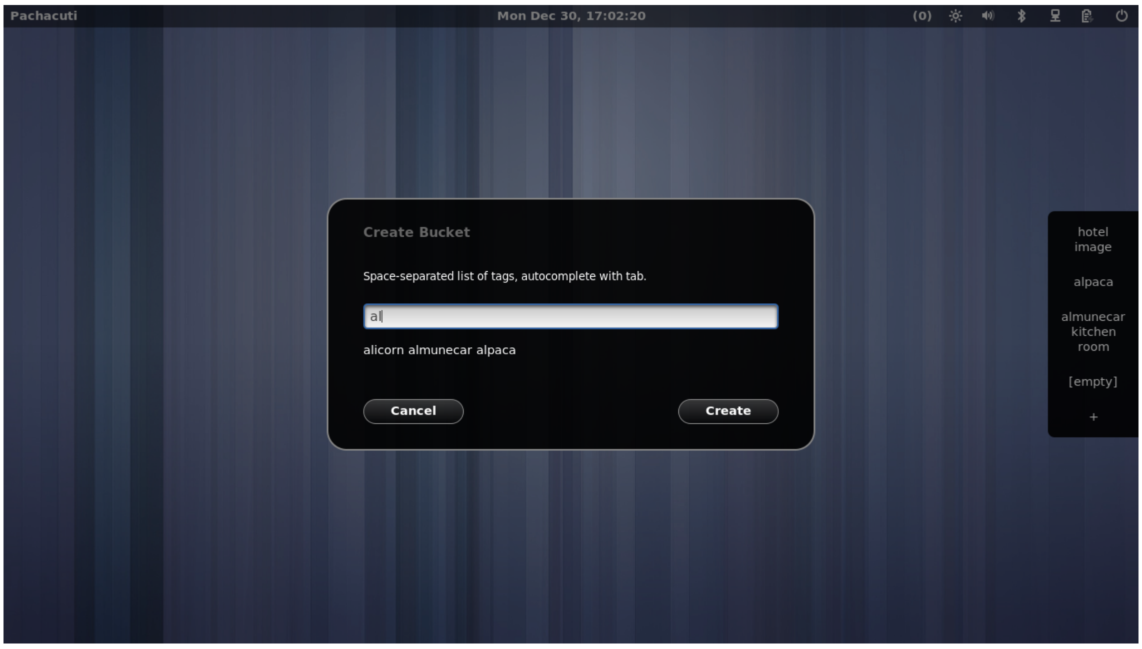


Figure 4.1: Panel for tagging and managing buckets.

As seen in figure 4.1, the panel consists of several lines of text. Except for "[empty]" (section 4.4) and "+", each of these lines represent a tag. The tags are also divided into several groups, each one defining a bucket. A bucket is simply a set of tags which provides the user with a way to both tag and browse content. The width and height of the panel changes depending on how much space the buckets require. The panel is however always centered on the right edge of the main monitor. To tag content, all the user has to do is to drop it into a bucket using drag-and-drop. The content will then be added to the tagging system and tagged with the tags defining the bucket.

Buckets can be created, modified, and removed by the user. To create a bucket the user must click on the plus symbol in the panel. The user will then be presented with the create bucket dialog (figure 4.2). This dialog asks the user to enter a space-separated list of tags which will define the bucket. In order to make it easier for the user to find the desired tags, he or she can use the autocomplete feature by pressing the tab key.



Space-separated list of tags, autocomplete with tab.

al

alicorn almunecar alpaca

Figure 4.2: Dialog for creating a bucket.

The autocomplete feature will look for tags with the same prefix as the tag currently being entered by the user. Up to ten of these tags are then presented as suggestions below the text entry field. This provides the user with hints about which tags are currently present in the system. The feature also looks for the longest common prefix of the suggested tags. The tag currently being entered by the user is then replaced by this prefix. In the case shown in figure 4.2 this would have no effect. However, if the user was to enter “alp” and press tab there would only be one tag left to suggest, which longest common prefix would be the suggested tag itself. The system would then replace “alp” with “alpaca” (plus a trailing space), saving the user some time and preventing typos. Had there also been a tag “alpaca_fiber” in the system, then the system would autocomplete with “alpaca” (without a trailing space) while showing both “alpaca” and “alpaca_fiber” as tag suggestions.

To remove a bucket from the panel the user can right-click on it and select “remove bucket”. This will only remove the actual bucket from the panel and does not affect any of the tagged content. If the user instead wishes to change the tags defining the bucket, he or she can right-click on it and select “modify tags”. Doing so will bring up a dialog similar to the create bucket dialog, but with the defining tags already present in the text entry field. Middle-clicking on a bucket will also open the modify bucket dialog.

## 4.4 Browser

To open a bucket the user simply needs to click on it. This brings up the browser (figure 4.3) in which the user can view previously tagged content. The user can close the browser by clicking on the desktop, or switch between buckets by clicking on them in the panel. The user can also create or modify buckets while the browser is open. The browser will automatically adjust to the width of the panel.

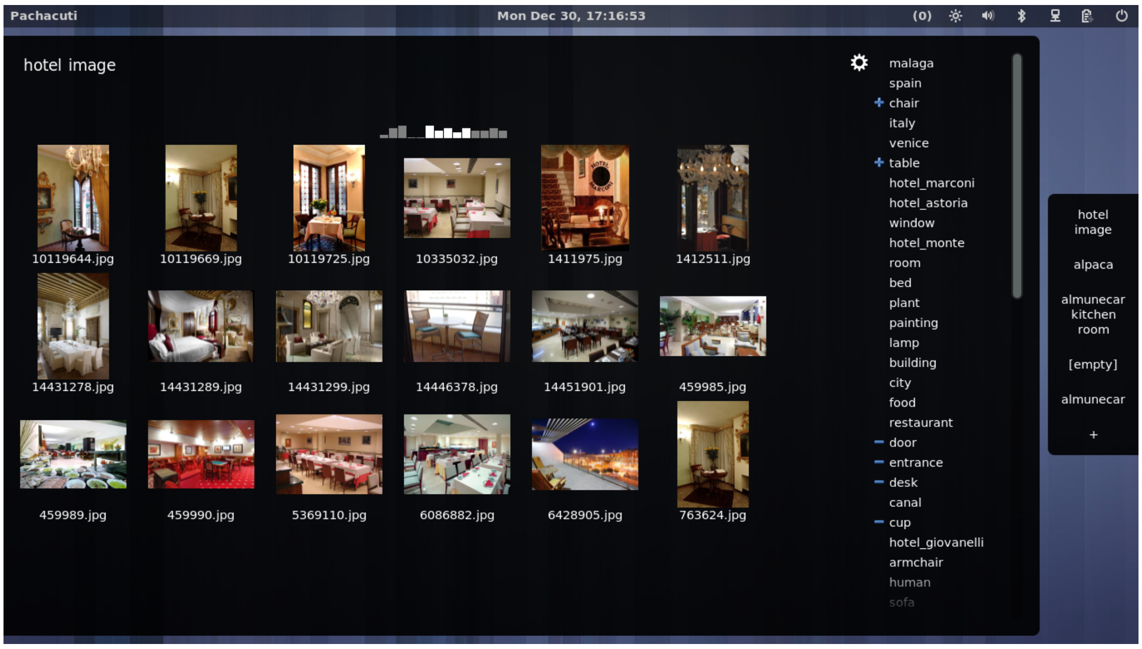


Figure 4.3: The bucket browser.

There are several different tools and utilities in the browser. In the top-left corner are a number of labels showing which tags define the currently open bucket. These labels can be picked up via drag-and-drop. If the user wants to remove a search criterion, he or she can just drop the label outside of the browser. This is an alternative to editing buckets via the modify bucket dialog.

Directly below the labels is the timeline which can be used to filter results by date. The timeline consists of a bar chart in which each bar represents a day. The height of each bar shows the relative number of items present for that day. The filter can be activated by clicking on one of the bars. Doing so will cause the bucket to only display results for that particular day. Multiple days can be selected by holding down the left mouse button and dragging the cursor across several bars. The filter is applied when the left mouse button is released. The current selection can be aborted by moving the cursor outside of the timeline area while still holding down the mouse button. To clear the filter the user can click on the empty area next to the bar chart.

A tooltip appears below the timeline filter when the cursor is hovering over a bar or when multiple bars are being selected. This tooltip informs the user of which dates his or her selection will include. Figure 4.3 has an active timeline filter. The bucket shown in the figure affects fourteen days in total, so fourteen bars are visible.

As indicated by their difference in brightness, the five bars in the middle have been selected by the user, thus limiting the results to those five days only.

The right-most part of the browser consists of the related tags filter. This filter lists all tags present for the items in the results except for those defining the bucket. The tags are ordered so that the higher up a tag is, the more items have that tag in common. The tags in the filter have three different states, namely “none”, “include”, and “exclude”. The none state is active by default and has no effect on the results shown in the bucket. The include state limits the results so that only items having that tag are shown. The exclude state is the opposite, hiding all items having that tag. Users can cycle through these states by right or left clicking on the tags. Figure 4.3 has some tags set to include as indicated by the plus symbol, and others set to exclude as indicated by the minus symbol. The rest of the tags have been left in their initial none state.

Below the timeline filter and to the left of the related tags filter is the results view. This view presents up to 256 of the items which match the bucket’s defining tags and all active filters. Clicking on an item will close the browser and open the item in its default application. Right-clicking opens up a context menu where the user can open the item, open the item’s containing directory, or open the item’s properties dialog (section 4.5). Middle-clicking on an item will also open the item’s properties dialog.

If the user wants to remove the tags defining a bucket from an item, he or she can simply drop the item outside of the browser area using drag-and-drop. Note that only the tags visible in the upper-left part of the browser will be removed from the item. Similarly, the user can add tags to items by using drag-and-drop together with the related tags filter. This is done either by dropping a dragged item onto a tag in the filter, or by dragging a tag from the filter and dropping it onto an item. Additionally, just like the user could remove tags defining a bucket by dropping the upper-left labels outside of the browser, the user can also add more tags by dropping tags from the related tags filter onto the labels. Dragging and dropping items and tags is mainly just a shortcut for editing buckets and items by text.

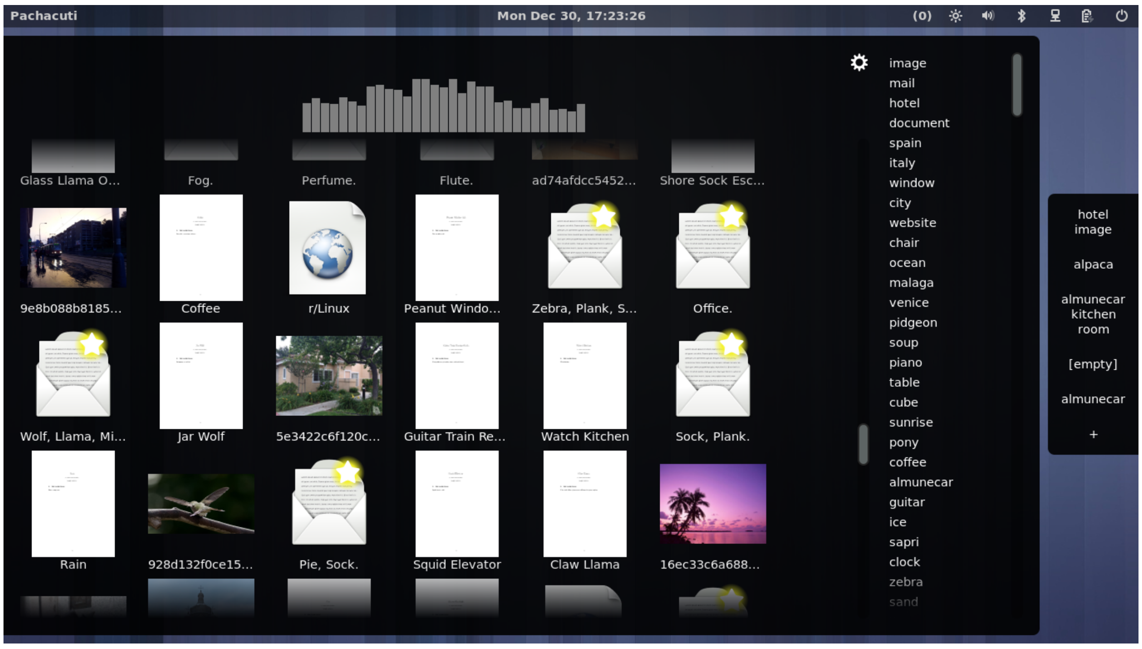


Figure 4.4: An empty bucket viewed in the browser.

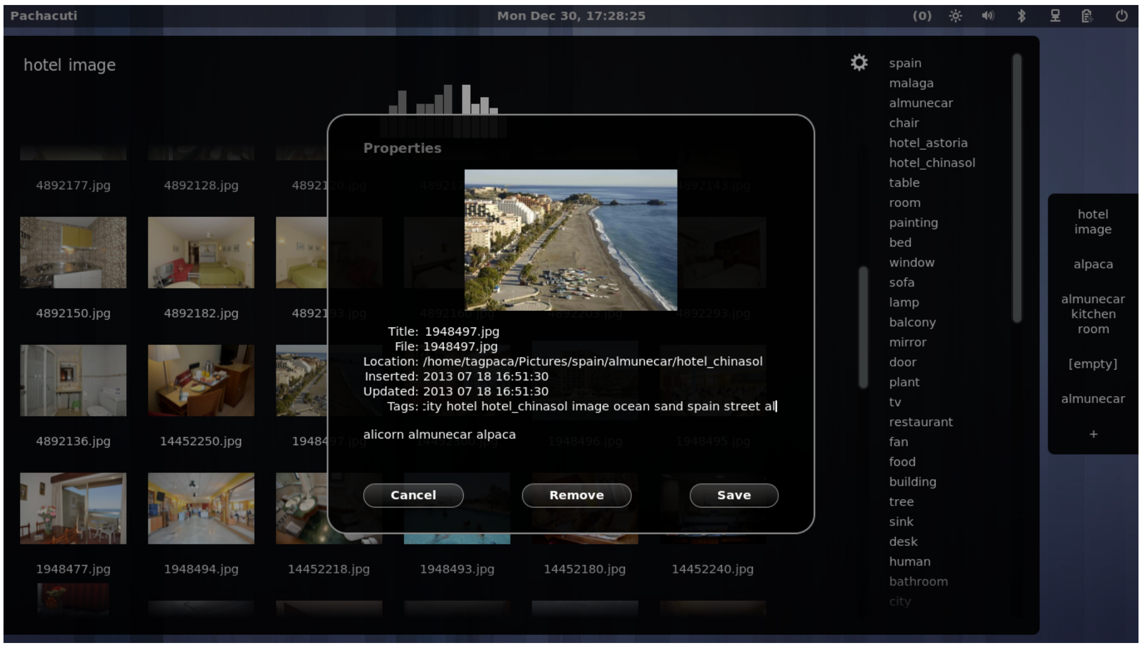
If a bucket is created without tags or if all tags are removed by editing a bucket, then the result is an empty bucket (figure 4.4). Empty buckets can be recognized in the panel as “[empty]”, but have no tag labels in the upper-left part of the browser. Empty buckets can be useful for finding previously used tags, especially if the user is interested in tags for items added on a specific date.

The reason buckets can be used for finding tags is that the two filters, timeline and related tags, not only filter the results, but also filter each other. For example, if the user selects a range of dates, then all tags in the related tags filter that have no items for that date range and are in the “none” filtering state will be hidden. Likewise, if the related tags filter is activated, then this will affect the number of items shown by the bars in the timeline filter. The goal is to give users hints about how their filtering affects the results and to make filtering easier by hiding unnecessary choices. The filters can of course be restored by making the other filter less restrictive.

Lastly, we have the bucket control menu which is accessible by clicking on the cog icon located above the top-right corner of the timeline filter. The control menu allows the user to clear all filters. Doing so resets both the timeline filter and the related tags filter. Note that the state of the filters is persisted between bucket switches and system restarts. Having a quick and easy way to restore buckets to their initial state is therefore important.

The control menu also allows the user to change if the timeline filter should affect the date on which an item was added to the tagging system, or if it should affect the most recent date on which an item's tags were changed. Additionally, the user can use the control menu to change the ordering of the results view. Items can be sorted by title, date added, or date updated in either ascending or descending order. The notes dialog (section 4.6) can also be opened using the control menu.

## 4.5 Properties



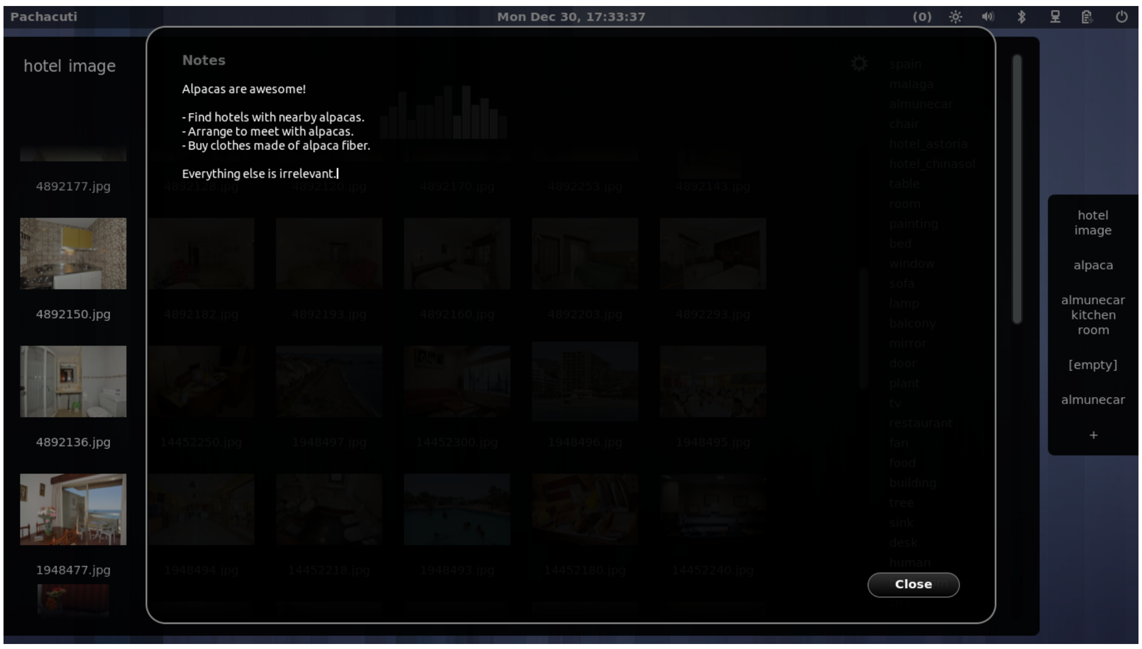
```
   Title: 1948497.jpg
    File: 1948497.jpg
Location: /home/tagpaca/Pictures/spain/almunecar/hotel_chinasol
Inserted: 2013 07 18 16:51:30
 Updated: 2013 07 18 16:51:30
    Tags: :ity hotel hotel_chinasol image ocean sand spain street al

alicorn almunecar alpaca
```

Figure 4.5: The item properties dialog.

The item properties dialog (figure 4.5) provides a more detailed view of items. Using the properties dialog the user can see an item's title, path, added date, updated date and attached tags. A thumbnail twice the size as those presented in the results view is also displayed. This makes it less likely that the user is forced to open the item in an external program. If some information is hidden due to the limited size of the dialog, then the user simply has to move the mouse cursor over that information to see it as a tooltip.

The user can also use the dialog to edit an item's title or attached tags via text fields. The title of an item is the text displayed in the results view. Just like with the create and modify bucket dialogs, tab completion can be used when modifying an item's attached tags. Tag suggestions are displayed below the attached tags text field. Changes can be reverted by clicking on the "Cancel" button or saved using the "Save" button. In addition, items can be removed completely from the tagging system by clicking on the "Remove" button. Note that this is very different from dropping the item outside of the browser via drag-and-drop as it affects all tags, not just those defining the bucket.

## 4.6 Notes



Alpacas are awesome!

- Find hotels with nearby alpacas.
- Arrange to meet with alpacas.
- Buy clothes made of alpaca fiber.

Everything else is irrelevant.

Figure 4.6: The notes dialog.

The notes dialog (figure 4.6) allows the user to write down notes related to a specific bucket. The dialog consist of one large text area which could be used as, for example, a to do list for a bucket related to a specific project. The text is saved when the dialog is closed, which happens when the user either presses the escape key or clicks the "Close" button. The notes are kept if the bucket is modified, but lost if the bucket is removed.

# 5 Implementation

This section describes the implementation of the tagging system. The general architecture is described, as well as important decisions and implementation details. This section is loosely based on the view template in *Documenting Software Architectures: Views and Beyond (2nd Edition)*[33].

Keep in mind that the TagPaca implementation is just a prototype of a tagging system and should not be used in a production environment. TagPaca was designed using an exploratory approach with only a general idea for its design. Additionally, most of the languages, libraries, and other tools used for the implementation were new to the author. The information in this section can however still be used to help guide the design and implementation of a similar tagging system.

## 5.1 Primary Presentation

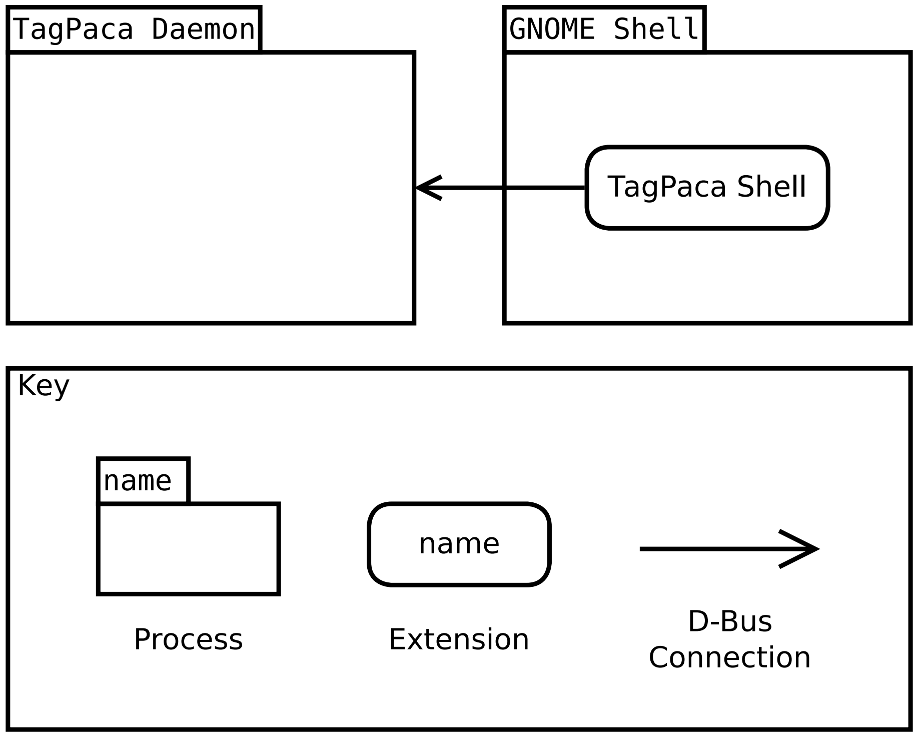


Figure 5.1: The general structure of the system.

The tagging system runs in multiple processes, each being either a stand alone program or an extension running within some other program. The main architectural idea was that extensions for several different programs should all connect to a single instance of the TagPaca daemon. The extensions should provide their host application with tagging support while the daemon provides the extensions with access to the tagging database.

## 5.2 Elements

This section describes the various elements present in the system. The technologies used to create these elements are described in more detail in section 5.9. The implementation conforms to the XDG Base Directory Specification[32].

### 5.2.1 TagPaca Daemon

TagPaca Daemon provides D-Bus services for accessing the tagging system's database. It was implemented in the Vala programming language which provides excellent support for implementing D-Bus services. The Daemon uses an SQLite database via the SQLHeavy wrapper library. The daemon provides no user interface, nor does it have any configuration options. Its only purpose is to provide other processes with access to tag-related services. TagPaca Daemon consists of 1623 lines of code, most of it being Vala. The GNU build system is used to build and install the daemon.

### 5.2.2 GNOME Shell

GNOME Shell is a compositing window manager, panel, and application launcher for the GNOME desktop environment. TagPaca requires a slightly modified version of GNOME Shell to function properly. The modification required by TagPaca is a more complete support for XDND, a protocol for drag-and-drop between applications in the X Window System. This addition simply allows GNOME Shell extensions to accept drag-and-drop drops from any GNOME application. GNOME Shell is written partially in C using the GObject object system, and partially in JavaScript running in Gjs. GNOME Shell extensions are mainly written in JavaScript, but can import and use GObject classes written in other languages. The modifications to GNOME Shell consists of a 537 lines long patch.

### 5.2.3 TagPaca Shell

TagPaca Shell is the extension running within GNOME Shell which provides users with the tagging system's graphical user interface. The extension was implemented in JavaScript because it is the primary language used for GNOME Shell extensions. TagPaca Shell both reads and writes data to the tagging database via a D-Bus connection to the TagPaca Daemon. St, Clutter, and GNOME Shell widgets are used to draw most of the graphical user interface, the exception being that Cairo is used to draw the timeline bar chart (section 4.4). TagPaca shell uses GSettings to store all its settings, including buckets with filter states and notes. The extension consists of 5737 lines of code, most of it being JavaScript.

## 5.3 Relations

This section describes the various relations present in the system. The technologies used to create these relations are described in more detail in section 5.9.

### 5.3.1 Extension

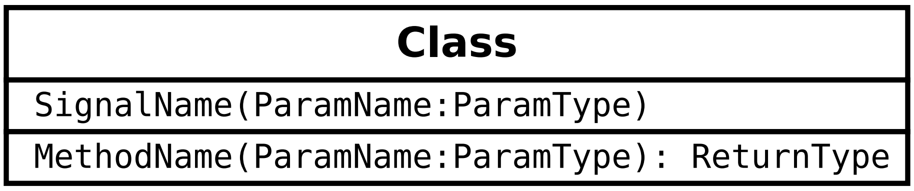
The extension relation describes code which adds additional functionality to some already existing software without directly modifying its source code. In this case the additional functionality is always support for interacting with the tagging system via the host application.

### 5.3.2 D-Bus Connection

The D-Bus connection relation describes inter-process communication. Clients (edge sources) can open connections to service providers (edge targets).

Client processes can, after opening a connection, call exported methods on class instances in the service provider process. Clients can also read from or write to class instance properties, as well as connect to class instance signals in order to receive events from the service provider. In this case the service provider is always the TagPaca Daemon, while the clients are extensions running within other processes.

## 5.4 Interfaces



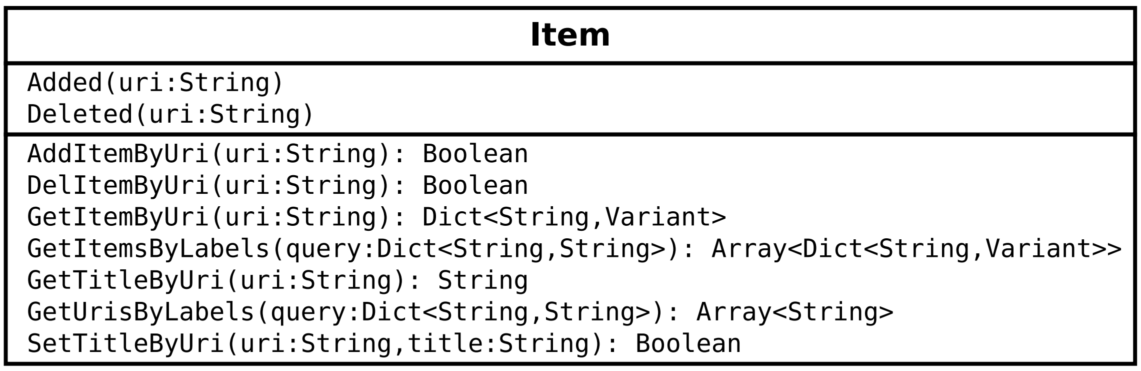
```
Class
SignalName(ParamName:ParamType)
MethodName(ParamName:ParamType): ReturnType
```

Figure 5.2: The key to the interface diagrams.

This section describes the D-Bus interfaces of the TagPaca Daemon. TagPaca Daemon does not provide any properties, but instead only signals and methods. The interfaces documented here are useful when constructing tagging extensions for programs. None of the internal interfaces are listed here, only those exposed via D-Bus. The key for the diagrams can be found in figure 5.2.

Some methods use a dictionary parameter called query. This parameter allows for dynamically selecting which filters should be applied to requests. The available filters are include, exclude, filter, order, and limit. Include and exclude both use a space-separated list of tags while filter, order, and limit use SQL syntax. It is important to be careful when using the filter, order, and limit options as they are inserted directly into the database query, thus making SQL injections possible. If anything, the use of such a hack should accentuate the fact that TagPaca is a prototype.

### 5.4.1 Item



```
Item
Added(uri:String)
Deleted(uri:String)
AddItemByUri(uri:String): Boolean
DelItemByUri(uri:String): Boolean
GetItemByUri(uri:String): Dict<String,Variant>
GetItemsByLabels(query:Dict<String,String>): Array<Dict<String,Variant>>
GetTitleByUri(uri:String): String
GetUrisByLabels(query:Dict<String,String>): Array<String>
SetTitleByUri(uri:String,title:String): Boolean
```

Figure 5.3: Item access via TagPaca Daemon.

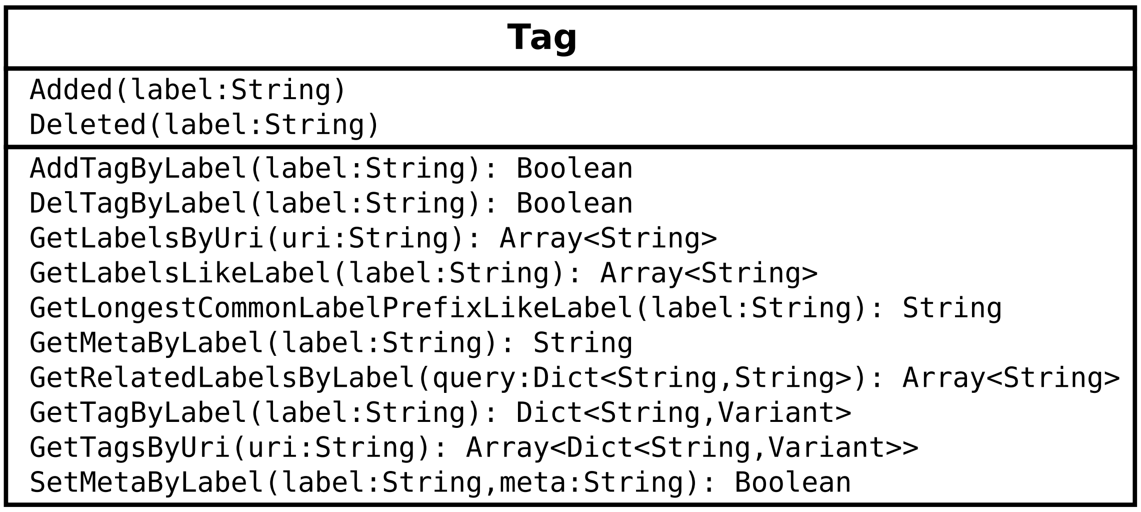
The Item interface (figure 5.3) provides item access via TagPaca Daemon. An item is simply a reference to some content such as a local file, e-mail message, or webpage address. Items are referenced by URI, each item having exactly one unique URI. A few examples can be found in table 5.1.

| **Type** | **URI** |
|---|---|
| File | file:///home/jocke/Pictures/alpaca.jpg |
| Directory | file:///home/jocke/Pictures/ |
| E-mail | tpmail://8bbd08c8bbc685f0043585a6b735df81 |
| Webpage | http://en.wikipedia.org/wiki/Alpaca |

Table 5.1: Example URIs for content.

The ID, title, date added, and date modified values of an item can be retrieved via its URI. It is also possible to find URIs by labels (tags), remove items, or change item titles. The date added and date modified values are set automatically and can only be changed using the Raw interface (section 5.4.5). Clients can request to be notified when an item is added or removed by connecting to the signals provided by the interface.

### 5.4.2 Tag



```
Tag
Added(label:String)
Deleted(label:String)
AddTagByLabel(label:String): Boolean
DelTagByLabel(label:String): Boolean
GetLabelsByUri(uri:String): Array<String>
GetLabelsLikeLabel(label:String): Array<String>
GetLongestCommonLabelPrefixLikeLabel(label:String): String
GetMetaByLabel(label:String): String
GetRelatedLabelsByLabel(query:Dict<String,String>): Array<String>
GetTagByLabel(label:String): Dict<String,Variant>
GetTagsByUri(uri:String): Array<Dict<String,Variant>>
SetMetaByLabel(label:String,meta:String): Boolean
```

Figure 5.4: Tag access via TagPaca Daemon.

The Tag interface (figure 5.4) provides tag access via TagPaca Daemon. A tag is referenced by its label, which is simply a string containing the tag name. The ID and meta data of a tag can be retrieved via its label. The meta data is currently not used for anything, but can be used by extensions to store any data in the JSON format. The interface provides a method for retrieving all tags belonging to a specific item. Methods for auto completion (getting the longest common prefix) and tag suggestions (getting related labels) are also provided. Clients can request to be notified when a tag is added or removed by connecting to the signals provided by the interface.

### 5.4.3 ItemTag

The ItemTag interface (figure 5.5) provides a way to attach and detach tags for items via TagPaca Daemon. There is also a method for checking whether or not an item has a specific tag. Clients can request to be notified when a tag is attached or detached by connecting to the signals provided by the interface.

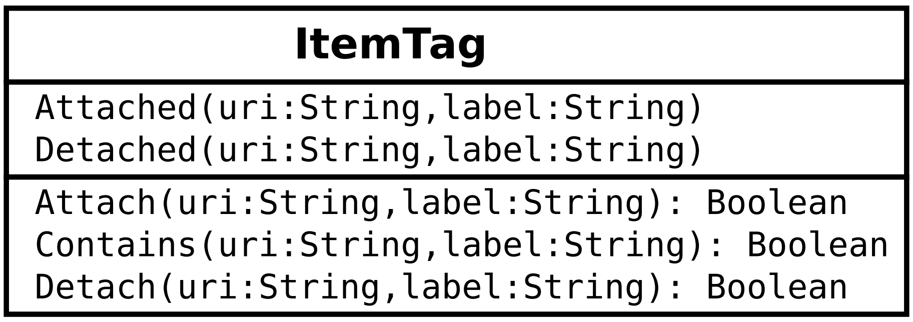

ItemTag
Attached(uri:String,label:String)
Detached(uri:String,label:String)
Attach(uri:String,label:String): Boolean
Contains(uri:String,label:String): Boolean
Detach(uri:String,label:String): Boolean

Figure 5.5: ItemTag access via TagPaca Daemon.

### 5.4.4 Stat

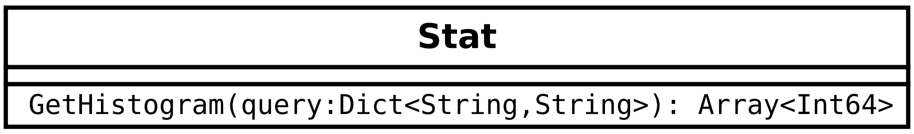


Figure 5.6: Stat access via TagPaca Daemon.

The Stat interface (figure 5.6) provides a way to retrieve statistics via TagPaca Daemon. The only supported operation is to retrieve an integer array containing interlaced time stamps and item counts. The array shows the number of items that were added or updated during specific time ranges. In other words, the array contains data useful for constructing histograms.

### 5.4.5 Raw

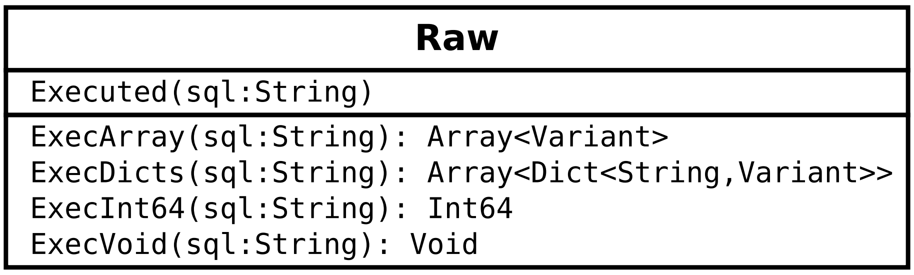

Raw
Executed(sql:String)
ExecArray(sql:String): Array<Variant>
ExecDicts(sql:String): Array<Dict<String,Variant>>
ExecInt64(sql:String): Int64
ExecVoid(sql:String): Void

Figure 5.7: Raw access via TagPaca Daemon.

The Raw interface (figure 5.7) provides a way to run SQL queries on the tagging database via TagPaca Daemon. The difference between the methods is the structure of the returned data. Clients can request to be notified when an SQL query has been successfully executed by connecting to the signal provided by the interface.

## 5.5 Behavior

TagPaca Daemon will be started automatically by D-Bus when a client attempts to use to it. Users therefore need not concern themselves with the daemon's life cycle. The daemon runs one instance per user but will never start unless actually needed. If a user kills their instance of the daemon, then it will stay inactive until a client once again needs it.

## 5.6 Variability

While this degree project only included TagPaca Shell, it is possible to add any number of user interfaces to the system. One example could be an extension for Nautilus which would allow users to tag files directly in the file manager. TagPaca Daemon can handle multiple clients simultaneously, so using multiple extensions at the same time is not an issue.

## 5.7 Rationale

The main architectural decision was to use separate processes for different parts of the system. Although inter-process communication makes the implementation more complex, it also makes it possible to add tagging support to several different applications while still allowing the user to access all of the tagged content from a single graphical user interface. The initial idea was to create extensions for multiple GNOME applications, primarily for those where tagging would be of interest. However, TagPaca Shell ended up being the only extension utilizing the daemon.

## 5.8 Challenges

This section describes the most important challenges the author encountered while developing the tagging system.

### 5.8.1 Drag And Drop

The initial idea was to allow users to tag content in multiple different GNOME applications via extensions. However, multiple extensions would have required far too much time and effort to implement. It was instead decided that tagging should be done via drag-and-drop in TagPaca Shell.

At first this seemed like an easy thing to accomplish. However, GNOME Shell did not completely implement any protocol for drag-and-drop interactions with external applications. It was possible to drag-and-drop widgets within GNOME Shell itself, but there was no way to accept a drop from another application.

To solve this, the drag-and-drop support in GNOME Shell had to be extended. GNOME Shell already had partial drag-and-drop support used for switching focus between external applications. For this GNOME Shell used the XDND protocol via xlib, so extending that seemed like a good idea.

Multiple parts of GNOME Shell written in both C and JavaScript had to be modified. The XDND protocol support had to be extended using xlib in C, while the API for accepting drops had to be extended using JavaScript. The extended API was designed to be very similar to how internal drops were handled, the only difference being additional methods for requesting and receiving drop data.

Solving this problem was crucial to the project. Had it not been solved, then tagging new content would have been very cumbersome in TagPaca Shell. Alternatively, the project could have reverted to the initial plan of tagging content through external programs.

### 5.8.2 Single Thread

GNOME Shell draws the desktop using a single thread, one which is also shared between all extensions. This means that if one extension performs a long-running task synchronously, then the whole desktop will freeze until that task is completed. Additionally, there is currently no way to create additional threads for extensions written purely in JavaScript. This in combination with that thumbnails take a while to load makes it difficult to keep the user experience smooth.

Initially, all thumbnails were loaded and added to the results view in one go. This caused the desktop to freeze for several seconds while opening a bucket. Because animations are based on timestamps, this also caused all animations to become slide shows containing only a couple of frames.

The solution was to load thumbnails one by one while the thread was otherwise idle. Queries to the TagPaca Daemon were made before animations were started, ensuring that potential delays would be difficult for users to notice. However, loading thumbnails only when the thread was idle significantly prolonged the total loading time. To counteract this, a simple system for caching and reusing thumbnail widgets was implemented. The cached thumbnails remain in memory until either the GNOME Shell process is stopped, or the maximum number of cached widgets has been reached.

Although using an additional thread to load thumbnails would have decreased the total loading times, the now implemented solution proved very effective. The desktop no longer froze when opening a bucket, and all animations were presented with a decent frame rate. It is however still possible to see some frame drops in programs displaying smooth animations while opening buckets.

## 5.9 Technologies

This section briefly describes the various technologies used to create the tagging system. All tools and libraries are free and open source software.

- **GObject** [16] is an object system for the C programming language. Many programming languages offer bindings for GObject which allows them to import and use GObject classes. There are many libraries providing GObject classes, all of which can all be used directly in any language with support for the object system. TagPaca Daemon consists only of GObject classes.

- **Vala** [27] is a programming language targeting the GObject object system. It has a syntax and style similar to C#, but does not run within a virtual machine. Instead, Vala is compiled to C which in turn is compiled to native machine code. Because Vala classes are actually GObject classes they can be used directly in all languages with GObject support. TagPaca Daemon was implemented in the Vala programming language.

- **GNU Build System** [18] is used by TagPaca Daemon for compilation and installation. The build system is responsible for ensuring that all required dependencies are available, for compiling all files with the proper tools in the proper order, and for placing executables and data in the correct locations during installation. The build system also generates data such as the D-Bus service file, ensuring that D-Bus knows where TagPaca has been installed.

- **SQLHeavy** [21] is a GObject wrapper for the SQLite C library. The library provides an object-oriented SQLite library for programming languages with GObject support. SQLHeavy also allows programs to listen for events related to the database using GObject signals. TagPaca Daemon uses SQLHeavy as its interface to the tagging database.

- **JavaScript** [19] is a programming language mainly used on websites. In this case it runs locally within the Gjs JavaScript engine. Gjs allows programmers to use GObject classes directly in JavaScript, thus making a large number of native libraries available in the language. TagPaca Shell was implemented in the JavaScript programming language.

- **GNOME Shell** [15] is the program providing users with the desktop user interface in the GNOME desktop environment. GNOME Shell is written partially in C and partially in JavaScript, relying on GObject and Gjs to bridge the gap between the two languages. TagPaca Shell was implemented as a GNOME Shell extension.

- **XDND** [2] is a protocol for drag-and-drop within the X Window System. The protocol allows programs to inform each other about what is being dragged, and whether or not they can accept the drop. GNOME Shell's XDND support was extended for this project in order to allow GNOME Shell extensions, such as TagPaca Shell, to accept drops from other GNOME applications.

- **Clutter** [4] is a GObject-based graphical user interface library which uses OpenGL for rendering. The library is used by GNOME Shell for drawing most of the desktop user interface.

- **St** [22] is a collection of graphical user interface widgets built on top of Clutter. The library is both shipped with and used by GNOME Shell. The widgets in St are also used by GNOME Shell extensions, such as TagPaca Shell.

- **Cairo** [10] is a library for vector graphics. Cairo has many uses, one example being that it can be used by GNOME Shell extensions to draw graphics. TagPaca Shell uses Cairo for drawing the timeline filter histogram (section 4.4).

- **D-Bus** [11] is a message bus for inter-process communication. It is used in multiple desktop environments via many different APIs and programming languages. TagPaca uses D-Bus for communication between the daemon and extensions.

- **GSettings** [17] is a centralized settings system for the GNOME desktop environment. It allows programs to load and store settings, as well as receive notifications when settings have been changed. TagPaca Shell uses GSettings for its settings and for persisting buckets.

# 6 Usability Study

This section describes the study identifying changes to the usability of the GNOME desktop environment. Results are interpreted in section 6.6 and their vailidity addressed in section 6.7.

## 6.1 Setup

The usability study performed in this degree project was loosely based on *Testing The Design With Users* in *Task-Centered User Interface Design*[1] by Clayton Lewis and John Rieman. This study did not follow all of their recommendations and used its own approach for comparing two different systems, TagPaca and GNOME applications, to one another.

The test users consisted of two different groups, the less experienced testers and the more experienced ones. The groups were used mainly to make it easier to reference the different testers in this report. The term “experience” in this case relates to how used a tester was to use new software systems. The less experienced testers were those who mainly use computers as tools for work. These testers generally only use a handful of applications and mostly stick to the same programs they have always used. The more experienced users were software developers. These testers use and explore many different new software systems for work, fun, or simply to satisfy their curiosity. Because they are software developers they also spend much time on building new software systems themselves. The reason for having selected these two groups was to see if a tester’s experience affected the usability of the two systems.

Six tasks (section 6.4) were created for the testers to solve. Applications (section 6.3) and an environment (section 6.2) to perform the tasks in were also needed. After performing the tasks, testers were asked five questions related to the study (section 6.5). The results have been summarized in section 6.6 and discussed in section 6.7.

Contrary to the recommendations in *Task-Centered User Interface Design*[1], testers were provided with limited assistance for both systems when necessary. The assistance included only things covered by the introduction to the study as well as the names of tags and directories. The reason was that testers, especially the less experienced ones, could not be expected to remember everything about a new desktop environment and five new applications after only a short introduction. Another reason was that directory and tag names are something rather personal. Two separate users are not likely to use the exact same tags and locations for their content. Still, testers had to make due with the structure of the environment for this study.

## 6.2 Environment

The testing environment contains images, documents, e-mails, website bookmarks, and browser history. Testers were presented with this environment and asked to carry out a number of tasks with both GNOME applications and TagPaca.

The goal of the environment was to simulate a user directory containing a large amount of well-organized content. The environment contains manually tagged and organized content which we call *targets*, as well as content which has been organized and tagged randomly called *distractors*. The number of items in the environment can be seen in figure 6.1.

| | Targets | Distractors | Total |
|---|---|---|---|
| **Images** | 216 | 500 | 716 |
| **E-mails** | 20 | 256 | 276 |
| **Documents** | 8 | 256 | 264 |
| **Web Pages** | 36 | 574 | 610 |
| **Total** | 280 | 1586 | 1866 |

Table 6.1: The number of items in the testing environment.

### 6.2.1 Targets

The manually organized and tagged content will serve as a targets for the tasks the testers will perform. To simplify the study a very specific topic was selected for this content, namely hotels for vacations. The content consists of pictures from various hotels downloaded from the Internet, links to various hotel and booking websites, forged e-mail conversations about hotels and the cities they are located in, and a few short PDF documents containing a few words about each hotel. Manually tagging the targets was tedious, showing the need for automatic tagging in TagPaca.

Figure 6.1 shows the timeline of the target content. The dates below the timeline correspond to days in July 2013. The dates gives a general idea of when information about each hotel appears to have been added from the perspective of the testers. Note that the dates are not very strictly specified. For example, content related to Hotel Marconi appears as if it had been added between the 13:th and the 15:th of July. The timeline was created mainly as a tool to help plan the environment. Testers were not shown the timeline before or during their testing sessions.

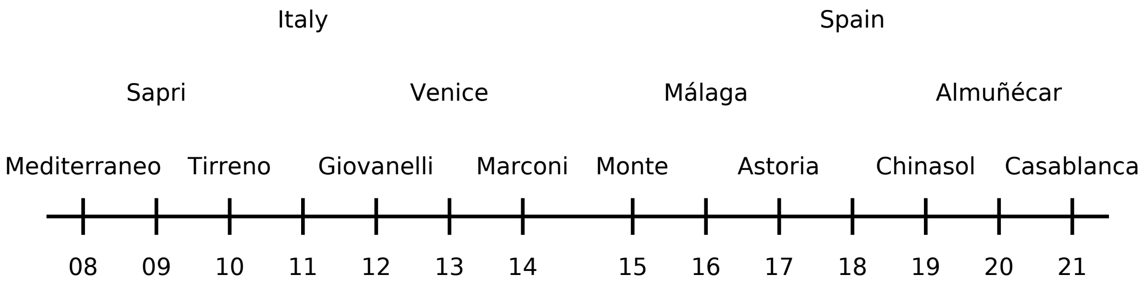


Figure 6.1: The general timeline of the enviroment data.

### 6.2.2 Distractors

Randomly organized content served as distractors in the usability study. The distractors were there to make the testers' tasks more difficult to solve, either by creating unwanted noise or confusing the tester. If there had only been content the user was supposed to find, then finding that content may not have been very difficult depending on the task. The distractors were generated using a set of one hundred English nouns, and contact information from six Swedish prime ministers.

Some of the random distractor tags had already been used properly together with the manually tagged content. This was done to force the testers to be more specific when searching for content. For example, if a tester were to look for pictures of chairs in hotels, then the tester must make sure to search for both chair *and* hotel. If the tester would search only for chair, he or she would be presented not only with pictures of chairs in hotels, but also with a large quantity of distractor images. This simulates the use of the same tag across multiple topics, which is how tags are likely to be used in a real world scenario.

A small shell script (figure 6.2) was created which, together with WebCollage[36] and GNU Wget[5], downloaded random images from the Internet. The downloaded images were manually inspected in order to find suitable distractors and to remove images which could haven been considered offensive. Out of over 2500 randomly downloaded images, 500 were deemed acceptable distractors.

```
webcollage --urls-only | while read url; do
        wget "$(echo $url | cut -d ' ' -f 1)"
done
```

Figure 6.2: Shell script for downloading random images from the Internet.

Another script was used to attach random tags to the randomly downloaded images. One downside to this was that the chosen tags had nothing to do with what was portrayed in the images. For example, a picture of a beach may have been tagged with the tag “forest”. However, this potential problem seemed worth the risk as the image distractors were there only to add noise to the environment. Tagging the distractors manually would have required quite a bit of effort which was better spent on more important issues.

Browser history, document, and e-mail distractors were all generated using the same set of tags as those used for images. The generated distractors did not suffer from the same problem as the images where tags had nothing to do with the actual content. Instead, the tags themselves became part of the generated content.

As an example, consider the generation of a distractor e-mail. First, a sender and time sent was chosen for the e-mail. Then, a set of up to ten tags were selected with which the e-mail would be tagged. The sender and tags were then reused to generate the body of the e-mail. Finally, up to four tags were picked from the already selected tags to create the e-mail’s subject header.

The browser bookmark distractors were an exception to this as they were chosen manually and tagged properly. The reason was that initially only browser bookmarks would haven been used in tasks related to finding web pages. This did not require a large enough number of distractors to justify creating another generator. However, browser history was later on also considered. This required item generation due to the large amount of web pages a user visits during the course of one month. Removing properly tagged distractors did not make sense, so they were kept alongside of the generated web page visits.

### 6.2.3 Initialization

Content generation and randomized tagging were all done with a collection of Python scripts. LaTeX was used by one of the scripts to create actual PDF documents from plain text.

Generating content and specifying tags is however not enough to create an environment for the testers. Somehow this information had to be inserted into both the tagging system and the GNOME applications. Simple TSV[24] files were used to store information about the content. An example to how these files were structured can be seen in table 6.2.

| uri | title | time | tags |
|---|---|---|---|
| http://graph.net/ | Graph | 2013-07-05 17:13:23 | website plate graph |
| http://moon.eu/ | Moon | 2013-07-19 07:46:16 | website stone moon owl |
| http://harp.com/ | Harp | 2013-07-09 17:37:06 | website candle balloon harp |
| http://pony.org/ | Pony | 2013-07-09 13:24:25 | website pony alpaca |
| http://coffee.se/ | Coffee | 2013-07-20 21:38:31 | website coffee cookie cat |

Table 6.2: Sample TSV file for the environment initialization.

The TSV files were either generated or created manually depending on what kind of content they specified. TSV files for browser bookmarks and target images were created manually. TSV files for documents, browser history, and distractor images were generated with random tags. Finally, TSV files for e-mails were generated using scripts which extracted information from mbox[34] files.

When all TSV files had been created they were read by other scripts which inserted the data into both the tagging system and the GNOME applications. This two-stage process ensured that the data could be manually edited and easily inspected before being inserted. This also allowed multiple initialization scripts to rely on the same TSV files.

Relationships between data files and initialization scripts can be seen in figure 6.3. In the graph, nodes with rounded corners represent scripts while the rest represent data files. The five nodes without children represent data files which were directly used by either the tagging system or the GNOME applications. The four data files without parent nodes were created manually by the author of this degree project, who also both wrote and ran the scripts.

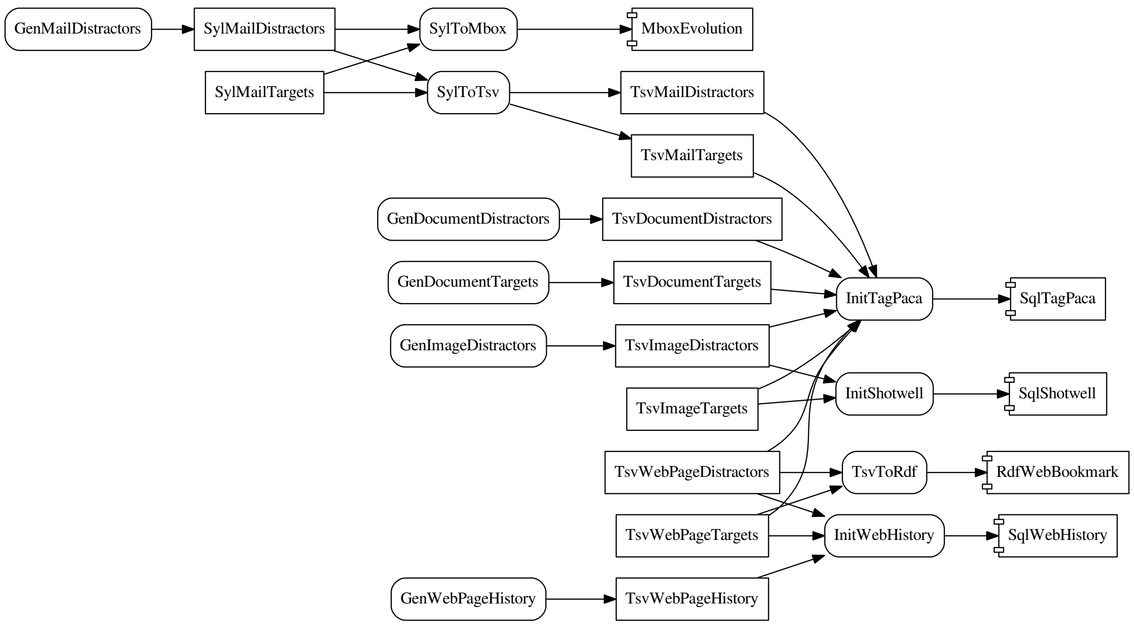


Figure 6.3: Data flow of scripts for creating the environment.

After creating the initialization scripts there was still one thing left to do before they could be used, namely setting up an empty environment. A new user account was created for the environment. This was done on the author's personal laptop on which the usability tests also were be performed. The GNOME applications and the tagging system were started once and then closed again. This was done in order to let the programs create their basic configuration and database files themselves, freeing the initialization scripts from that responsibility.

The initialization scripts populated SQLite databases for both the tagging system and the GNOME applications. The scripts also adjusted the atime (time last accessed) and mtime (time last modified) timestamps for files in the file system. Additionally, the scripts generated an RDF file containing a list of bookmarks for the web browser.

## 6.3 Applications

Along with the desktop shell, four GNOME applications have been included in the usability study. The programs selected for the study specifically manages images, websites, documents, and e-mails. The applications have, as described in section 6.2, been populated with both target and distractor content.

### 6.3.1 Nautilus

GNOME's file browser (figure 6.4) is called Nautilus[9]. Nautilus provides a simple graphical user interface for browsing both local and remote file systems. The program also provides a basic search functionality for locating files and directories by name and type.

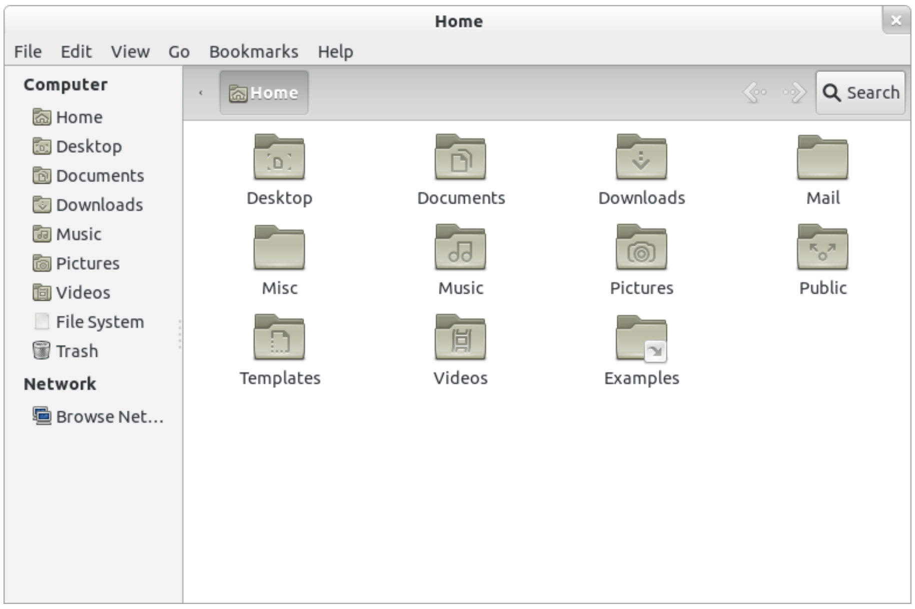
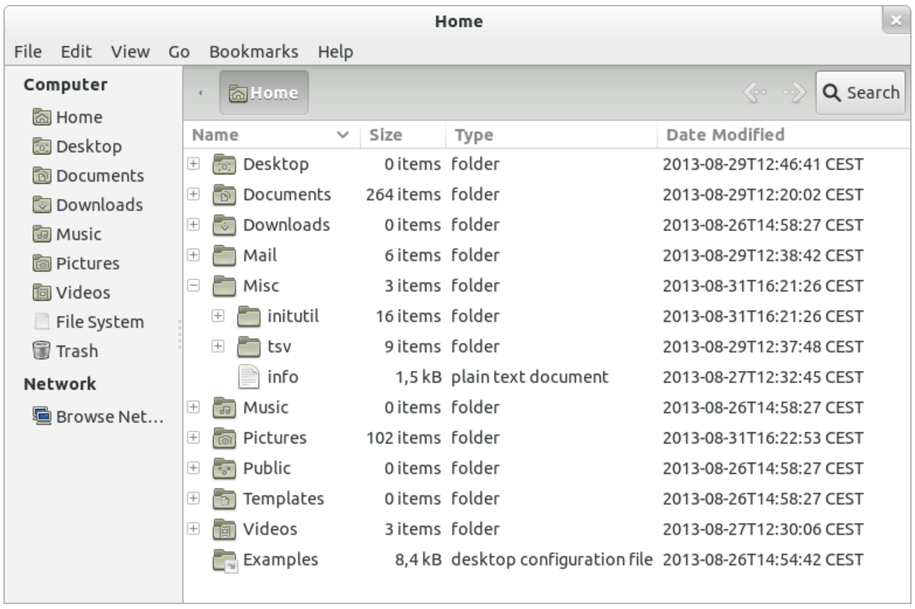


Figure 6.4: Nautilus, the GNOME file browser.

One task required testers to locate files based on their modification date. Nautilus' search features does not allow users to limit the results based on this value. However, testers could still complete the tasks either by looking at the properties of files or by switching to another view. Nautilus' list view (figure 6.5) presents files and directories as a single table. There, the testers could manually go through the results to find the relevant files.

Nautilus features a plug-in system where additional functionality can be added to the program without modifying its source code. It may be possible to extend Nautilus' search functionality to allow users to limit the results by file modification date. This was not attempted as the study was aimed at testing GNOME as is, the



| Name | Size | Type | Date Modified |
|---|---|---|---|
| Desktop | 0 items | folder | 2013-08-29T12:46:41 CEST |
| Documents | 264 items | folder | 2013-08-29T12:20:02 CEST |
| Downloads | 0 items | folder | 2013-08-26T14:58:27 CEST |
| Mail | 6 items | folder | 2013-08-29T12:38:42 CEST |
| Misc | 3 items | folder | 2013-08-31T16:21:26 CEST |
| initutil | 16 items | folder | 2013-08-31T16:21:26 CEST |
| tsv | 9 items | folder | 2013-08-29T12:37:48 CEST |
| info | 1,5 kB | plain text document | 2013-08-27T12:32:45 CEST |
| Music | 0 items | folder | 2013-08-26T14:58:27 CEST |
| Pictures | 102 items | folder | 2013-08-31T16:22:53 CEST |
| Public | 0 items | folder | 2013-08-26T14:58:27 CEST |
| Templates | 0 items | folder | 2013-08-26T14:58:27 CEST |
| Videos | 3 items | folder | 2013-08-27T12:30:06 CEST |
| Examples | 8,4 kB | desktop configuration file | 2013-08-26T14:54:42 CEST |

Figure 6.5: The list view in Nautilus.

only exception being the tagging system itself. Still, the plug-in system would no doubt be useful for future integration with the tagging system, perhaps in a manner similar to what the Tracker plug-in (section 3.1) already does.

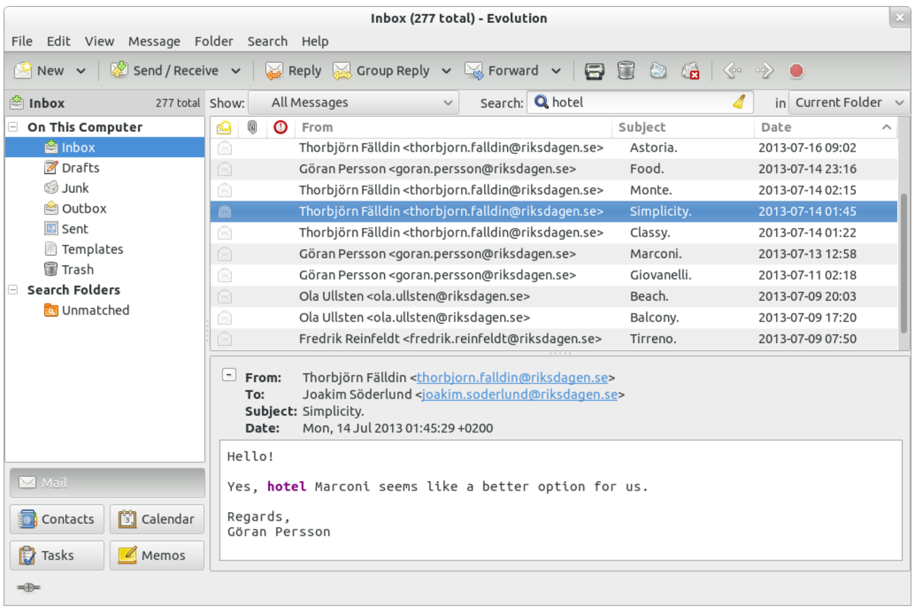
Nautilus 3.4 was used in the usability study. This version was outdated when the tests were performed as the next major release was targeted for many large changes[35]. One notable feature in the new version was an improved search functionality. Newer Nautilus versions can, for example, search within documents instead of only for their file names. This is something that could have affected the results of one of the tasks (section 6.4). Nautilus 3.4 was used regardless as it was the version provided by the Linux distribution used in the tests.

Before performing the tasks in the study, testers were shown basic navigation through directories, how to search for content, and how to switch between the different views.

### 6.3.2 Evolution

The e-mail client used in the GNOME desktop environment (figure 6.6) is called Evolution[8]. Evolution provides a table view listing of e-mails along with an area for reading mail within the same window. It also offers the possibility to open mail in separate windows by double-clicking on them. Additionally, Evolution provides calendars, address books, task lists, and memos. However, the only feature covered by the usability study was the e-mail client.

Evolution has a simple search feature in which it is possible to search for e-mails by text. This includes not only searching for senders or subjects, but also searching through the body of messages. Evolution also has a much more advanced search feature. Using the advanced search, users could for example search for an e-mail with a specific sender, without a certain person receiving a carbon copy, with one specific word in the subject, excluding another word from the message body, while having been sent on a specific day. Advanced searches can optionally be saved for later use in the form of search folders. Testers only needed to use the simple search



| From | Subject | Date |
|---|---|---|
| Thorbjörn Fälldin <thorbjorn.falldin@riksdagen.se> | Astoria. | 2013-07-16 09:02 |
| Göran Persson <goran.persson@riksdagen.se> | Food. | 2013-07-14 23:16 |
| Thorbjörn Fälldin <thorbjorn.falldin@riksdagen.se> | Monte. | 2013-07-14 02:15 |
| Thorbjörn Fälldin <thorbjorn.falldin@riksdagen.se> | Simplicity. | 2013-07-14 01:45 |
| Thorbjörn Fälldin <thorbjorn.falldin@riksdagen.se> | Classy. | 2013-07-14 01:22 |
| Göran Persson <goran.persson@riksdagen.se> | Marconi. | 2013-07-13 12:58 |
| Göran Persson <goran.persson@riksdagen.se> | Giovanelli. | 2013-07-11 02:18 |
| Ola Ullsten <ola.ullsten@riksdagen.se> | Beach. | 2013-07-09 20:03 |
| Ola Ullsten <ola.ullsten@riksdagen.se> | Balcony. | 2013-07-09 17:20 |
| Fredrik Reinfeldt <fredrik.reinfeldt@riksdagen.se> | Tirreno. | 2013-07-09 07:50 |

**From:** Thorbjörn Fälldin <thorbjorn.falldin@riksdagen.se>
**To:** Joakim Söderlund <joakim.soderlund@riksdagen.se>
**Subject:** Simplicity.
**Date:** Mon, 14 Jul 2013 01:45:29 +0200

```
Hello!

Yes, hotel Marconi seems like a better option for us.

Regards,
Göran Persson
```

Figure 6.6: Evolution, the GNOME e-mail client.

feature in order to solve the tasks, but were not prevented from using any of the more advanced features.

While Evolution has built-in support for POP, IMAP, and SMTP, additional protocols can be added via plug-ins. The plug-in system is however not limited to mail protocols and could be used for future integration with the tagging system. For example, users could be allowed to tag e-mails for the tagging system directly in the e-mail client.

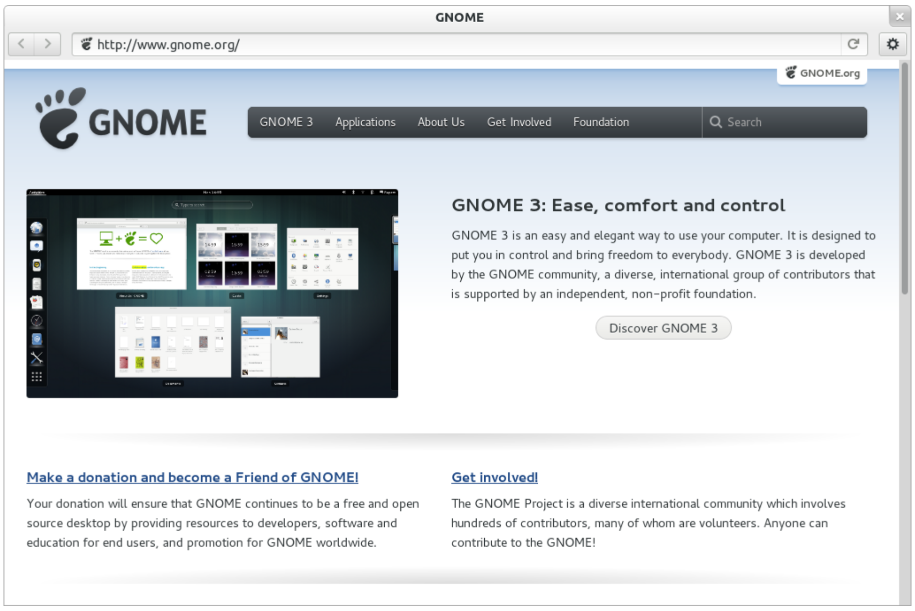
Evolution also supports labeling e-mails, a feature which was not used in the usability study. Admittedly, this could affect the result of the study as one feature for content organization was unavailable. There were two reasons for not applying labels to e-mails in the environment. The label system appears to have been designed for broad subjects, not easily supporting the large number of tags used in the tagging system. It was not possible to use the simple search to browse for labels like users can in Shotwell (section 6.3.4). Instead, it was only possible to search for a single label at a time by selecting one from a combo box. The combo box listed all labels present in the system within a single drop-down menu. Users could browse multiple labels using the advanced search feature. The selection, however, still only allowed for labels to be added one by one via combo boxes. The second reason was the time it would have taken to write scripts for inserting the data into Evolution's database. Thus, this particular feature was left neglected.

Sylpheed[23], an e-mail client which is not part of the GNOME desktop environment was for some time planned to be used in place of Evolution. The reason was that the functionality for opening a specific e-mail in Evolution using another program had been removed[3]. Evolution still ended up being used in the GNOME part of the tests. However, Sylpheed was left responsible for showing e-mail messages opened via the tagging system.

Before performing the tasks in the study, testers were shown how to read e-mails, and how to use the simple search functionality.

### 6.3.3 Web

GNOME Web (figure 6.7), previously known as Epiphany, is a basic WebKit-based web browser for the GNOME desktop environment[7]. GNOME Web supports a few common browser features such as bookmarks, history, and extensions. The browser is basic in the way that it does not have many features other than those previously listed. Of course, additional features can be added via extensions.



GNOME 3: Ease, comfort and control

GNOME 3 is an easy and elegant way to use your computer. It is designed to put you in control and bring freedom to everybody. GNOME 3 is developed by the GNOME community, a diverse, international group of contributors that is supported by an independent, non-profit foundation.

Make a donation and become a Friend of GNOME!

Your donation will ensure that GNOME continues to be a free and open source desktop by providing resources to developers, software and education for end users, and promotion for GNOME worldwide.

Get involved!

The GNOME Project is a diverse international community which involves hundreds of contributors, many of whom are volunteers. Anyone can contribute to the GNOME!

Figure 6.7: Web, the GNOME web browser.

For the usability study the most interesting aspects of the browser were the history and bookmark features. The tests focused on finding content that had been seen before, such as previously visited websites. While all the target websites had been bookmarked, it was not possible to see when a bookmark had been added. Instead, testers were forced to find out when a website had last been visited via the history. Both the history window and the bookmarks window have simple text-based search features. This allowed the testers to filter out content they were not interested in.

Before performing the tasks in the study, testers were shown how to find and use both the bookmarks and the history features.

### 6.3.4 Shotwell

GNOME's photo organizer (figure 6.8) is called Shotwell[20]. Shotwell supports browsing and searching for photos by event, tag, or file name. While this program may focus more on personal photographs taken by the user rather than those downloaded from the Internet, it was still used in the study because it provides a way for testers to search for images by tags.

Shotwell indexes photos across multiple directories in order to create a single searchable collection. To build this collection Shotwell utilizes information gathered from image meta data, along with file and directory names. Duplicates are eliminated by comparing the checksums of imported images. Additionally, the meta data of images may actually contain tags defining their content. Such tags are recognized by Shotwell and are automatically added to the program's photo database.

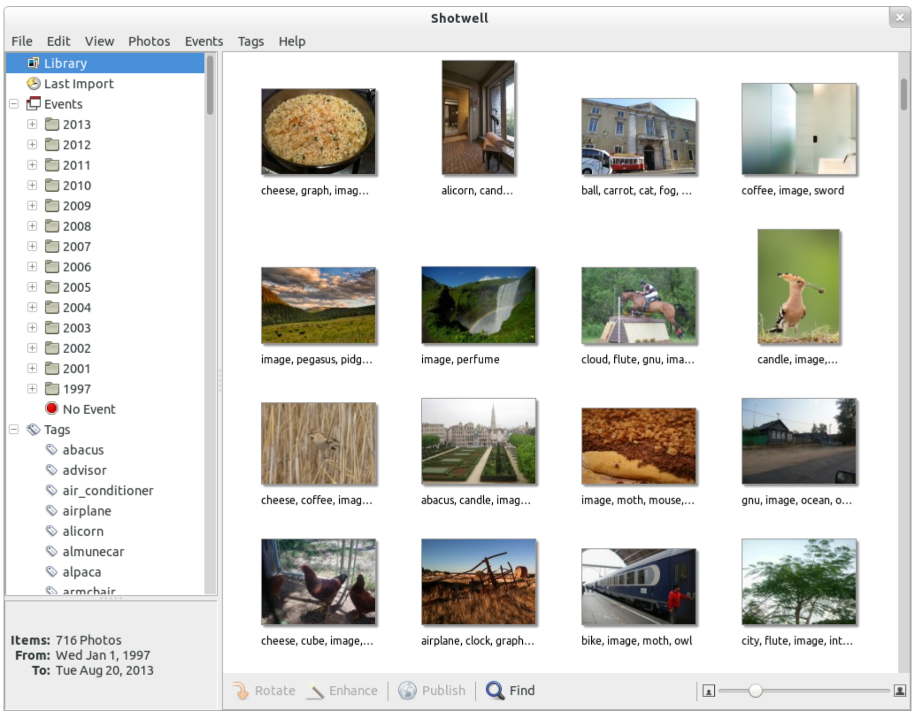


Figure 6.8: Shotwell, the GNOME photo organizer.

The events feature allows users to browse photos by date. The feature utilizes the meta data stored in images for checking when a photo was taken. Events are created automatically by date but can also be renamed, merged, or created manually. There is however no way of browsing or searching for images by when they were added to the collection. Browsing by date added would have been very useful for the testers when they performed the usability study tasks. Changing the time meta data of images to match the date added value in TagPaca was therefore considered. However, the idea was rejected as users are not at all likely to do so themselves after downloading an image.

When indexed by Shotwell some distractors were discovered to have predefined tags in their meta data. These tags were removed from Shotwell and replaced by the same tags as those present in the tagging system. This was done so that all images in the environment had the exact same tags regardless of what program they were viewed in.

Before performing the tasks in the study, testers were shown how to browse and search for images. How tags and events work in Shotwell was also explained.

### 6.3.5 TagPaca

See section 4 for information about how the tagging system is used. Before performing the tasks in the study, testers were shown how to create buckets, use tab completion, open buckets, close buckets, switch buckets, filter items by date, filter items by tag, edit buckets by drag-and-drop, edit buckets by text, opening an item, viewing an item's properties, and changing the ordering of the results view.

## 6.4 Tasks

This section describes the tasks performed by the testers, each subsection corresponding to one task. The topic for all but one task was related to finding a hotel for a vacation. A short story was told about each task in order to help the testers understand what they were supposed to do. Before that, the overarching scenario for the study was told:

> You are planning a trip together with a few friends and were assigned the task of picking a hotel. During July 2013 you researched a few hotels in various cities, all of which were recommended by your friends. Now you would like to look back at the information you have previously collected.

Testers were themselves required to decide when they had completed the tasks. This was done so that the testers did not get hints that would lead them to the correct answer. Additionally, if a user was confident that an incorrect answer was correct, then it would hint at a problem with either the system or the task.

Although they required some time and effort, the tasks were designed to be pretty easy to solve in both TagPaca and the GNOME applications. To solve a task testers did not need to open any content, but were free to do so if they wanted to. The only exception was the task about finding documents (section 6.4.6). Testers would have had to open multiple documents if they were to solve that task using the GNOME applications.

### 6.4.1 Finding Mail

The testers were given the following task:

> You would like to know who recommended certain hotels to you and what people had to say about them. You decide to look through your mail in pursuit of this knowledge.
>
> A. Find all mail about the hotels located in Sapri (table 6.3).
>
> B. Find all mail about the hotels located in Venice (table 6.4).

| Tool | Tester | Total |
|---|---|---|
| Evolution | 1 | 1 |
| Evolution | 3 | 2 |
| TagPaca | 2 | 4 |
| TagPaca | 4 | 4 |
| Solution | - | 4 |

Table 6.3: The number of items each tester found in task 1A.

This task was mainly included to help the testers get a feel for the desktop environment, Evolution (section 6.3.2), and TagPaca. The task also shows how the tagging system performs when compared to a single application. That is, if the tagging system can be of any help even when users know exactly which program to use to find some content.

| **Tool** | **Tester** | **Total** |
|---|---|---|
| Evolution | 2 | 2 |
| Evolution | 4 | 2 |
| TagPaca | 1 | 4 |
| TagPaca | 3 | 4 |
| Solution | - | 4 |

Table 6.4: The number of items each tester found in task 1B.

There was also an unexpected point to this task, namely that searching by text for e-mails may not be enough. For example, an e-mail mentioning something specific about a hotel may not always include the name of the city in which the hotel is located. Such e-mails would therefore not be included in the results when searching only for the location.

This is confirmed by the results in tables 6.3 and 6.4. When using Evolution, no tester was able to find all e-mails requested by the task. Consider subtask A which requested testers to find all e-mails about the hotels in Sapri as an example. Testers only performed searches for “sapri”, “hotel sapri”, or similar. To find all e-mails about the hotels in Sapri the testers should also have performed searches for the names of the hotels they found in the search results. Additionally, one tester only recognized one e-mail as being about the hotels in Sapri despite the search returning three results, two of which were requested by the task.

All testers managed to find all of the requested e-mails in TagPaca despite not performing searches for hotel names. This shows that if e-mails are properly tagged, then users will be able find them even if the search phrase is not actually contained within the messages. Labeling the e-mails in Evolution may have helped if testers had been shown how to perform advanced searches.

The number of steps users had to take to reach their results were largely the same in both systems. The more experienced testers did not have any problems performing the task in any of the systems. However, one was less confident in the search results presented by Evolution. The tester performed additional searches and inspected a few related and unrelated items before going back to the initial search phrase and announcing the completion of the task. The tester did not do this when browsing the results of the tag-based system.

Although the task took much more time to complete, the results were largely the same for the less experienced users. The tester who found only one e-mail spent a very long time getting familiar with the Evolution search functionality and going through all e-mails related to hotels before, like one of the more experienced users, going back to the initial search results and announcing the completion of the task. This tester managed to find the correct e-mails quickly in TagPaca, but had to open and inspect the messages before being satisfied. It appeared as if the tester did not quite trust either system to return only relevant results.

The second less experienced tester had more trouble with the tagging system than Evolution. How the filtering of items worked did not appear clear to this tester. While finding the correct results on first try, the tester opted to apply additional tag filters using the related tags filter. After a couple of minutes of experimenting with the related tags filter, the tester announced the completion of the task. The tester seemed much more confident in the results returned by Evolution, despite not finding all of the requested e-mails.

### 6.4.2 Finding Images

The testers were given the following task:

> After looking through the mail you decide you want to know a little bit more about the hotels. What the hotels look like is important to you. Therefore you decide to take a look at the images you have saved.
>
> A. Find all images of the hotels located in Sapri (table 6.5).
>
> B. Find all images of the hotels located in Venice (table 6.6).

| **Tool** | **Tester** | **Total** |
|---|---|---|
| Shotwell | 1 | 48 |
| Shotwell | 3 | 48 |
| TagPaca | 2 | 48 |
| TagPaca | 4 | 48 |
| Solution | - | 48 |

Table 6.5: The number of items each tester found in task 2A.

| **Tool** | **Tester** | **Total** |
|---|---|---|
| Shotwell | 2 | 59 |
| Shotwell | 4 | 59 |
| TagPaca | 1 | 59 |
| TagPaca | 3 | 59 |
| Solution | - | 59 |

Table 6.6: The number of items each tester found in task 2B.

Like with the previous task, this task was designed to be help testers get a feel for the systems and to see how the tagging system compares to individual programs. In this case TagPaca was compared to the photo manager Shotwell (section 6.3.4) which supports tagging of images and videos. One interesting aspect of this task is that both systems support tags.

As can be seen in tables 6.5 and 6.6, all testers were able to find all the requested images for both tasks. The two more experienced users performed this task swiftly in TagPaca. One opted to create a new bucket while the other reused the bucket from the previous task, replacing the tag “mail” with “image”. The tester who previously doubted the results in Evolution did so again in Shotwell, performing multiple searches and then returning back to the initial search.

The two less experienced users were much more confident in the results returned by TagPaca this time. The two testers solved the task quickly, neither having to inspect the returned images before announcing its completion. Both users reused the bucket created during the previous task. One user opted to replace the “mail” tag with “image” while the other just removed the “mail” tag and instead used the related tags filter.

When using Shotwell one of the less experienced users simply used the search function to find the requested images. The other tester used a combination of the tag list and search function. After a few almost successful attempts the tester stopped using the tag list and found the correct results via the text search. However, the tester did not trust the results returned by Shotwell and decided to also search for images using the file manager. Not finding anything after a few attempts, the tester announced the completion of the task using the images found in Shotwell.

### 6.4.3 Finding Content By Place

The testers were given the following task:

> You were not happy with those hotels, so you decide to look at some other hotels as well. This time you are interested in all content directly related to specific hotels. This includes mail, images, documents, and websites that you have visited before.
>
> A. Find all content directly related to Hotel Casablanca (table 6.7).
>
> B. Find all content directly related to Hotel Astoria (table 6.8).

| Tool | Tester | Mail | Img | Doc | Web | Total |
|---|---|---|---|---|---|---|
| GNOME | 1 | 2 | 14 | 1 | 4 | 21 |
| TagPaca | 3 | 2 | 14 | 1 | 4 | 21 |
| TagPaca | 2 | 2 | 14 | 1 | 4 | 21 |
| TagPaca | 4 | 2 | 14 | 1 | 4 | 21 |
| Solution | - | 2 | 14 | 1 | 4 | 21 |

Table 6.7: The number of items each tester found in task 3A.

| Tool | Tester | Mail | Img | Doc | Web | Total |
|---|---|---|---|---|---|---|
| GNOME | 2 | 3 | 36 | 1 | 4 | 44 |
| GNOME | 4 | 3 | 36 | 1 | 4 | 44 |
| TagPaca | 1 | 3 | 36 | 1 | 4 | 44 |
| GNOME | 3 | 3 | 36 | 1 | 4 | 44 |
| Solution | - | 3 | 36 | 1 | 4 | 44 |

Table 6.8: The number of items each tester found in task 3B.

Testers were now expected to have at least a little bit of experience in dealing with both TagPaca and GNOME applications. This task was designed to test how TagPaca compares to multiple GNOME applications at the same time. That is, comparing a system which handles multiple content types to having one program per content type.

Once again, all testers managed to find all the requested items. The observant reader will however notice a mistake in one of the tests by looking at tables 6.7 and 6.8. The subtasks for tester 3 had accidentally been swapped by the author, causing TagPaca to be used three times in subtask A and the GNOME programs three times in subtask B. This was not believed to have affected the study enough to justify invalidating all results for that tester, especially considering that all testers managed to find all the requested items in both TagPaca and GNOME.

For the two more experienced testers, the results were as expected. Using TagPaca both testers managed to swiftly locate all the requested items by creating a bucket with nothing but the tag for the hotel. Neither tester bothered to take a close look the results. Instead, they quickly announced the completion of the task.

The more experienced testers used multiple different GNOME programs to solve the task. One tester used four programs while the other used three, the difference being that the latter used Nautilus instead of Shotwell to find images. It took a while longer for both testers to find the requested items using the GNOME applications simply because they had to use different programs for different item types. Both testers seemed confident that they had found what they were looking for with both the tagging system and the GNOME applications.

The two less experienced users used the GNOME programs in a similar manner. Again, one tester used four programs while the other used three, the difference being that the latter used Nautilus instead of Shotwell to find images. One tester also opted to use the address field to find previously visited websites, something which all other testers used the history window for. The two less experienced testers also seemed confident that they had found what they were looking for.

Using the tagging system for this task was more difficult for the less experienced testers. The first tester quickly found all the requested items, but used the related tags filter to pick out each content type one by one before reporting the results. While this is fine, the tester did not at first understand the that enabling more filters just added more constraints to the results. Thus, the tester attempted to enable filters for two different content types at the same time, expecting both types to be shown but being presented with nothing. After being reminded of the *and* relationship between filters, the tester proceeded by locating the various content types one by one.

The second less experienced tester had even more trouble figuring out how to solve this task. When creating a bucket the tester assumed that the system showed all content that he or she had asked for, expecting an *or* relationship between tags instead of the *and* relationship actually used by the system. The tester created a single bucket using multiple content type tags at once. Thus, the tester ran into the same problem as the other less experienced tester. Because there is no item which is a document, an image, and an e-mail at the same time the returned results were empty. The tester then started guessing various buttons for a while. Even after being reminded of the *and* relationship between tags the tester did not fully grasp that adding additional tags further restricted the search results. The tester then created one bucket each for a couple of different content types. After opening such a bucket the tester explained that he or she had been looking for the related tags filter which was now visible. Like the previous tester, this tester at first attempted to activate more filters to find more content. Although it took much longer than using the GNOME programs, the tester did in the end manage to solve the task by searching only for content types and enabling related tags filters.

### 6.4.4 Finding Hotels

The testers were given the following task:

> While those hotels seemed decent, you are interested in hotels with some very specific qualities. The problem is that you do not know which hotels have what you want. You are not interested in browsing for new hotels on the Internet.
>
> A. Find out which hotels have a sauna (table 6.9).

B. Find out which hotels have a kitchen (table 6.10).

| Tool | Tester | Total |
|---|---|---|
| Shotwell | 1 | 1 |
| Shotwell | 3 | 1 |
| TagPaca | 2 | 1 |
| TagPaca | 4 | 1 |
| Solution | - | 1 |

Table 6.9: The number of items each tester found in task 4A.

| Tool | Tester | Total |
|---|---|---|
| Shotwell | 2 | 1 |
| Shotwell | 4 | 1 |
| TagPaca | 1 | 1 |
| TagPaca | 3 | 1 |
| Solution | - | 1 |

Table 6.10: The number of items each tester found in task 4B.

The goal of this task was to see whether or not testers could use the systems to find meta data. That is, using the systems not to find actual content, but to extract additional information. As seen in tables 6.9 and 6.10, all testers managed to complete this task in both systems.

The two more experienced testers had no problem carrying out this task using TagPaca. They easily found the relevant content and then quickly figured out that they could see the names of the hotels they were looking for in the related tags filter. When using the GNOME applications both testers eventually found what they were looking for by using the tags in Shotwell. However, they had to inspect the tags of each image in the results individually before they came up with their answer. Additionally, one of testers decided to search through e-mails in Evolution before using Shotwell.

The two less experienced users did much of the same. However, they did not realize that they could find all relevant hotels in the related tags filter. Instead they opted to inspect the tags of each element in both TagPaca and Shotwell. The testers had some problems figuring out how to see which hotel a certain item belonged to in both TagPaca and the GNOME applications, but managed to figure that out eventually. In addition to Shotwell, one of the testers decided to look for more hotels using the other three GNOME programs, going back and forth between them before announcing the completion of the task.

### 6.4.5 Finding Content By Date

The testers were given the following task:

> You usually discuss hotels for the trip with your friends. However, one of them was away for a few days and missed your discussions. This friend would like to know everything about what he missed. This includes mail, images, documents, and websites that you have visited.

A. Find all content directly related to hotels for the 10:th, 11:th, and 12:th of July (table 6.11).

B. Find all content directly related to hotels for the 17:th, 18:th, and 19:th of July (table 6.12).

| **Tool** | **Tester** | **Mail** | **Img** | **Doc** | **Web** | **Total** |
|---|---|---|---|---|---|---|
| GNOME | 1 | 1 | 0 | 2 | 5 | 8 |
| GNOME | 3 | 1 | 3 | 2 | 6 | 12 |
| TagPaca | 2 | 1 | 44 | 2 | 6 | 53 |
| TagPaca | 4 | 1 | 44 | 2 | 6 | 53 |
| Solution | - | 1 | 44 | 2 | 6 | 53 |

Table 6.11: The number of items each tester found in task 5A.

| **Tool** | **Tester** | **Mail** | **Img** | **Doc** | **Web** | **Total** |
|---|---|---|---|---|---|---|
| GNOME | 2 | 2 | 0 | 1 | - | 3 |
| GNOME | 4 | 2 | - | 1 | 5 | 8 |
| TagPaca | 1 | 2 | 57 | 1 | 6 | 66 |
| TagPaca | 3 | 2 | 57 | 1 | 6 | 66 |
| Solution | - | 2 | 57 | 1 | 6 | 66 |

Table 6.12: The number of items each tester found in task 5B.

In this task the systems were tested for how easy it is to find content by date. More specifically, a range of three days when content of a specific topic should have been added. As seen in tables 6.11 and 6.12, all testers found all content when using TagPaca, but none succeeded when using the GNOME programs. A dash in the table signifies an invalid result where the tester picked a large number of unrequested items as his or her answer.

The four testers had no trouble swiftly completing the task using TagPaca. However, using the GNOME programs to solve it proved difficult. Regardless of experience, testers often jumped between multiple programs several times before finding what they were looking for. When they did find something, testers had usually looked for one content type at a time, searched for "hotel", and then manually scrolled through the results to pick out the relevant items.

It was made clear to the testers that they could use any GNOME program to find any type of data. Still, all users focused on attempting to find images via Shotwell despite being told that the program indexed images by date shot and not by date added. One of the more experienced users managed to find an advanced search feature in Shotwell which the author was unaware of.

The advanced search feature supported specifying multiple date ranges and search phrases. Still, the date ranges only affected the date when the picture was shot. Shotwell returned all 216 photos of hotels despite having been limited to a relatively small date range. The reason for this was that the tester had overlooked a combo box specifying how multiple search options related to one another. Thus, the tester searched for images with the tag "hotel" *or* images shot during the specified date range. Looking back at task 3 in section 6.4.3, this hints at that more experienced users assume an *and* relationship between search terms while less experienced users assume an *or* relationship.

In the end, no tester managed to solve the task of locating images by date added. The solution was to use the file manager to inspect directories containing the images of hotels and look for their modification dates. The tester who used the advanced search feature in Shotwell explained that he or she had thought of this, but wanted to find a prettier solution to the problem. Had the task instead been to find images with respect to when they were shot, then it would have been solvable with Shotwell but impossible to solve using TagPaca.

As seen in the tables, sometimes a tester would neglect to include a single website. For subtask A this was simply a case of overlooking an item when scrolling through a large number or results. In subtask B one of the hotel-related websites lacked the word "hotel" in both the URL and title of the page.

### 6.4.6 Finding Documents

The testers were given the following task:

> After discussing the hotels with your friend you decide to grab a cup of coffee at a nearby café. While there, you talk about all sorts of things. After a while you start talking about camelids, something you happen to know quite a bit about. You end up promising to send all the documents you have on this subject to your friend when you get back home.
>
> A. Find all documents directly related to alpacas (table 6.13).
>
> B. Find all documents directly related to llamas (table 6.14).

| **Tool** | **Tester** | **Total** |
|---|---:|---:|
| Nautilus | 1 | 3 |
| Nautilus | 3 | 3 |
| TagPaca | 2 | 15 |
| TagPaca | 4 | 15 |
| Solution | - | 15 |

Table 6.13: The number of items each tester found in task 6A.

| **Tool** | **Tester** | **Total** |
|---|---:|---:|
| Nautilus | 2 | 3 |
| Nautilus | 4 | 3 |
| TagPaca | 1 | 10 |
| TagPaca | 3 | 10 |
| Solution | - | 10 |

Table 6.14: The number of items each tester found in task 6B.

This task was included simply to show that file and directory names may not be enough for finding content. The task should be seen as a bit of a side note as it was specifically designed to show something, namely one advantage of searching for tags compared to file paths.

All testers completed the task relatively quickly, the less experienced users requiring a little bit more time than the more experienced ones. All testers also appeared confident in their findings regardless of which system they used. However, as tables 6.13 and 6.14 show, testers found only a fraction of the requested documents when searching for file names only. When the testers instead searched using tags they quickly found all the requested documents.

Systems which can search within documents would also have been able to find all the requested documents. As explained in section 6.3.1, this would have been the case if the study had used a more recent version of Nautilus. The same holds true for indexers such as Tracker (section 3.1).

Although all testers completed the task relatively quickly, one of the less experienced testers ran into a couple of problems. After creating a bucket in TagPaca the tester simply forgot to open it, just staring at an empty desktop for a while. The tester would been presented with the correct results by clicking on the bucket, but interpreted the empty desktop as if there were no results for his or her query. The tester then performed a few incorrect searches, this time not forgetting to open the bucket. Eventually the tester returned to his or her initial and correct search before announcing the completion of the task. The same tester also made a similar mistake while searching in Nautilus, neglecting to press enter after entering the search phrase. Like before, the tester eventually figured out what the problem was before announcing the completion of the task.

## 6.5 Questions

The testers were asked a few questions about the tagging system after having performed the tests. Testers were not told about any test results before answering the questions. This means that testers did not know whether or not they had found all of the requested items. The testers were asked to be absolutely honest when answering these questions. However, all testers personally know the author so a certain bias towards the tagging system is to be expected.

- How do you think the tagging system affected the usability of the desktop environment?

All testers thought that the tagging system improved the usability of the desktop environment by making content easier to find. Two of the testers were very enthusiastic about the tagging system while another was a little bit hesitant, pointing out the need to learn more about the system before being able to use it properly. The slightly hesitant tester also pointed out that the system helped the most when searching for items by date.

The remaining tester pointed out that the tagging system definitely helped during the tests. However, the tester also pointed out that if he or she had been the one to tag and organize the content the results may have been different. Though the tester still believed that such a tagging system would have been helpful in any case.

- Would you actively use a similar tagging system if it tagged content automatically?

Here all testers answered that were certain they would use such a system actively if they did not had to tag content themselves. One tester even went as far as saying that he or she would use the tagging system exclusively when searching for local content if this was the case.

- Would you actively use a similar tagging system if you had to tag content manually?

Surprisingly, all testers would still consider using the tagging system even if they had to spend time on tagging content manually. One of the less experienced testers stated that a requirement for using the system without automatic tagging would be that attaching tags to items was easy.

The two more experienced testers both stated that their use of the system would be very limited without automatic tagging. While they would still use it, they would use it more or less like a bookmarking system, tagging only important content. One of the more experienced testers also expressed concerns that the system could become useless if he or she would neglect tagging some of the important content.

- Do you have any suggestions for improvements to the tagging system?

By far the most common suggestion was to add a better way of finding out which tags are available in the system. Multiple testers suggested text filtering or sorting options for the tags in the related tags filter. Another tester wanted the create bucket dialog to suggest tags without requiring the user to press tab. One of the more experienced testers wanted to replace the current empty bucket feature with a new static button located next to the create bucket button. The new button would then always open an empty bucket, not requiring users to create them manually.

One of the more experienced testers wanted many improvements to the results view, such as adding a list view as an option. The tester also wanted the ability to select multiple items and perform operations such as cut, copy, and paste on them. Another suggestion was proper support for content type filters with abilities such as searching only for images of a specific size and file format. The tester's last suggestion was to utilize the meta data within images to perform some basic automatic tagging.

- Do you have any other comments or questions?

One of the more experienced testers was interested in what the initial idea of the project was (section 1) and how users could attach tags to content (section 4). Another tester said that he or she would need some time to learn and get used to such a system. One of the testers had only one comment, "I want it".

## 6.6 Results

The tests performed during this usability study points to that the tagging system did improve the usability of the desktop environment. There are three reasons for this, the first being that all testers managed to find all requested items when using TagPaca while missing several when using the GNOME applications. The second reason is that testers in most cases required an equal or less number of steps for completing tasks when using TagPaca compared to when using the GNOME applications. Lastly, all testers gave more or less positive feedback about the tagging system and stated that it did improve the usability of the desktop environment.

## 6.7 Discussion

While the results of this study points to that the tagging system did improve the usability of the desktop environment, the quality of the study is of course debatable. Although not designed to, the tasks and environment may have given the tagging system an unfair advantage.

One very good example is the task for finding content by date (section 6.4.5). Because TagPaca does not support browsing images by date shot and Shotwell not by date added, this task will either favor one system or the other. In this case it made the task impossible to solve using Shotwell, forcing testers to use Nautilus instead. The task was still solvable, but when looking for images TagPaca had been given a clear advantage.

Another thing that could have affected the results was the testing environment (section 6.2). The environment was set up in way similar to how a very meticulous user might store his or her data. Tags that had been attached to content manually had been verified several times to ensure that all tags of importance to the tasks were there. Had this been a real environment with manual tagging, then the user would no doubt have forgotten to attach several important tags to the content. This would have affected the number of items returned by TagPaca when the user performed a search for that content.

Because the file system is hierarchical, there is also no good way to map tags to paths in the file system. However, the files and directories were named in such a manner that a search for the key tags in each task would yield the correct results. The task for finding documents (section 6.4.6) is an exception to this as it was designed to show an advantage to tags over file names.

Another thing worth considering is user experience. If the testers had used both systems for a few weeks before performing the tasks, then the results of this study might have looked very different. The testers would no doubt have had an easier time using both systems if that had been the case.

In some tests we could clearly see the value of having one test group with more experienced users and another with less experienced ones. The more experienced testers had no trouble solving the tasks using TagPaca. However, both of the less experienced testers ran into trouble figuring how to use the tagging system. Neither of them had any prior experience with tagging. To them the tagging system provided an entirely new concept for organizing content in a computer. The more experienced users had run into tagging several times on various websites prior to participating in the study.

Finally, the last thing worth considering is the number of testers. Due to time constraints the study only included four. While this could greatly affect the quality of the study, it should still provide a general idea of the usability of the systems.

The information gathered about how the testers used the system, especially when they were struggling with the completion of tasks, can be used to improve the tagging system's user interface. This information also points out where special attention is needed for manuals or training when new users are introduced to a tagging system. The improvements mentioned by the testers during the questions session (section 6.5) should also be taken into consideration. Sadly, because of time limitations, such improvements must be left as future work.

# 7 Conclusion

The goal of this master degree project was to find out whether or not a highly integrated system for universal tagging of content improves the usability of a desktop environment. The usability study showed that such a tagging system made it easier for users to find content they were looking for. Making content easier to find should no doubt count as an improvement to the usability of a desktop environment. In other words, the highly integrated system for universal tagging of content did improve the usability of the desktop environment.

However, there is no definite way to directly measure usability. There is also no way to be sure that the study itself was equitable. Thus, although the usability study shows an improvement when using the tagging system, the conclusion should be taken with a grain of salt.

# 8 Future Work

The most important aspect missing in this degree project is automatic tagging. Tagging content manually requires much time and effort. Even tagging the relatively small number of target images for the usability study required quite a bit of effort (section 6.2). This is also reflected by the testers' answers to the questions in the usability study (section 6.5). The testers were more enthusiastic about using a tagging system with automatic tagging than one without.

The tagging system developed during this degree project was a prototype created mainly for the usability study. There is much work to be done on both its design and implementation before it can be distributed to end users. Effort could also be spent on trying to get such a tagging system bundled with some desktop environment.

Section 3 mentioned other projects attempting to make content easier for users to find. Instead of focusing on a tagging system, effort could be spent on providing these projects with improved user interfaces. Another idea would be a tagging system similar to that developed in this degree project, but using Tracker (section 3.1) as the back end.

# 9 References

[1] Task-centered user interface design. http://hcibib.org/tcuid/tcuid.pdf, August 2004.

[2] Drag-and-drop protocol for the x window system. http://www.newplanetsoftware.com/xdnd/, July 2010.

[3] Evolution - drop support for command-line email uris. https://git.gnome.org/browse/evolution/commit/?id=974924a8b8ceb3cabd6a4e517b1a1a1285fa137f, May 2011.

[4] Clutter project. http://blogs.gnome.org/clutter/, October 2012.

[5] Gnu wget. http://www.gnu.org/software/wget/, September 2012.

[6] Tracker. https://live.gnome.org/Tracker/, March 2012.

[7] Apps - epiphany - gnome wiki. https://wiki.gnome.org/Apps/Evolution/, August 2013.

[8] Apps - evolution - gnome wiki. https://wiki.gnome.org/Apps/Evolution/, August 2013.

[9] Apps - nautilus - gnome wiki. https://wiki.gnome.org/Apps/Nautilus, August 2013.

[10] Cairo graphics. http://www.cairographics.org/, August 2013.

[11] D-bus. http://www.freedesktop.org/wiki/Software/dbus/, May 2013.

[12] Gjs - gnome wiki. https://wiki.gnome.org/Gjs, July 2013.

[13] Gnome - about us. http://www.gnome.org/about/, June 2013.

[14] Gnome shell extensions - what's this? https://extensions.gnome.org/about/, June 2013.

[15] Gnomeshell - gnome wiki. https://wiki.gnome.org/GnomeShell, March 2013.

[16] Gobject reference manual. https://developer.gnome.org/gobject/, September 2013.

[17] Gsettings. https://developer.gnome.org/gio/stable/GSettings.html, August 2013.

[18] Introducing the gnu build system. http://www.gnu.org/savannah-checkouts/gnu/automake/manual/html_node/GNU-Build-System.html, June 2013.

[19] Javascript tutorial. http://www.w3schools.com/js/, September 2013.

[20] Shotwell - yorba. http://www.yorba.org/projects/shotwell/, September 2013.

[21] Sqlheavy - gobject sqlite wrapper - google project hosting. http://code.google.com/p/sqlheavy/, September 2013.

[22] St reference manual. https://developer.gnome.org/st/, September 2013.

[23] Sylpheed - lightweight and user-friendly e-mail client. http://sylpheed.sraoss.jp/en/, August 2013.

[24] Tab-separated values. http://en.wikipedia.org/wiki/Tab-separated_values, August 2013.

[25] Tracker source code repository. https://git.gnome.org/browse/tracker/, June 2013.

[26] Tweener documentation and language reference. http://hosted.zeh.com.br/tweener/docs/en-us/, September 2013.

[27] Vala - gnome wiki. https://wiki.gnome.org/Vala, August 2013.

[28] Zeitgeist. https://launchpad.net/zeitgeist, May 2013.

[29] Nepomuk. http://userbase.kde.org/Nepomuk, January 2014.

[30] Tag pictures so they're easier to find. http://windows.microsoft.com/en-us/windows/tag-pictures-easier-find, March 2014.

[31] Tags help you organize your files. http://support.apple.com/kb/HT5839, January 2014.

[32] Waldo Bastian, Ryan Lortie, and Lennart Poettering. Xdg base directory specification. http://standards.freedesktop.org/basedir-spec/basedir-spec-0.7.html, January 2012.

[33] David Garlan, Felix Bachmann, James Ivers, Reed Little, Judith Stafford, Len Bass, Paul Clements, Paulo Merson, and Robert Nord. Documenting software architectures - views and beyond, October 2010.

[34] Erik Hall. The mbox media type. http://tools.ietf.org/html/rfc4155, September 2005.

[35] William Jon McCann. Nautilus - cross cut. http://blogs.gnome.org/mccann/2012/08/01/cross-cut/, August 2012.

[36] Jamie Zawinski. Webcollage - exterminate all rational thought. http://www.jwz.org/webcollage/, January 2011.