Desktop Searches with Kat

BUSY KAT

The Kat desktop search tool turns up more than text strings.

BY ROBERTO CAPPUCCIO

The desktop search engine offers new and interesting opportunities for users to access information on their own computers. This new breed of tools lets you search your own computer in the same way you would search the Internet.

It comes as no surprise that the first tool of this category was created by Google. Another giant of the computer industry, Apple, has gone beyond this concept, integrating its own desktop search engine, called Spotlight, into the operating environment. In this setting, people do not need to know they are using a search tool.

About a year ago, I thought it was about time to equip Linux with an advanced desktop search tool, so I started developing Kat (Figure 1). Of course, I was aware of tools like *locate* or *find* that perform searches in the file system. Unfortunately, these tools lack some of the important features of a modern desktop search utility: they are only able to dig for information inside simple text-related files, they don't consider meta data, and their indexes need to be rebuilt manually. Moreover, they only look for information in files. Contemporary computer systems contain lots of information that is outside the file system, such as mail addresses, emails, contacts, play lists, and so on. We set out to create a tool that searches all the possible sources and even considers relationships between bits of information.

Developing Kat

The conventional notion of a search tool is a program that looks through files for a match to an alphanumeric text string. Most people are aware that a modern desktop search engine is a bit more exotic, but to truly appreciate the complexity of a tool like Kat, you must consider some of the complications implicit in the design. Kat must retrieve relevant information from a massive set of data, taking into consideration not only the names of the files, but also their contents and meta data. For images, we must collect a thumbnail, so that users will have an easier time browsing inside the possibly huge set of retrieved images. We also want to track the relationships between the various pieces of information. Because Kat must process many different types of data, it must be capable of reading many different file formats. As you'll learn in the following sections, the struc-

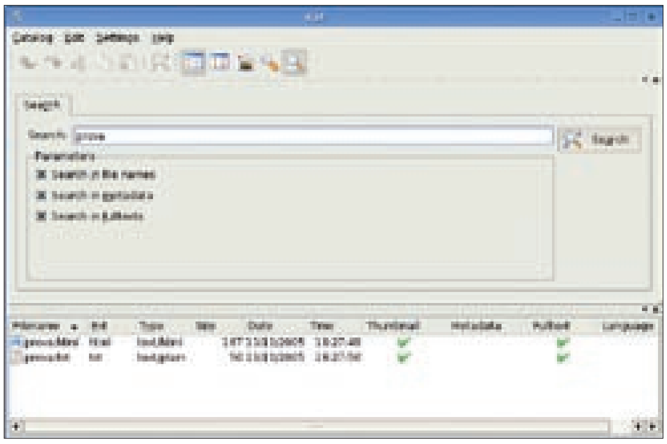


Figure 1: The Kat search utility lets you search for file names, metadata, file text, and more.

ture of the Kat search engine reflects the objectives of this design.

Indexer

The first step in building Kat was the design of the indexer. The indexer consists of a loop traversing all the folders contained in the base directory specified by the user. Kat lets users decide which part of the disk they want to index. We strongly suggest that each user index their own home directory, although other choices are allowed.

In order to extract as much information as possible, the indexer needs a way to cope with the different formats in which the files are saved. For text documents, the file format might be something like PDF, DOC, PS, or another format that does not save the text in a readable form. Depending on the format, sometimes the text is compressed, sometimes it is split into blocks, and sometimes it is written in encoded form.

A modular approach is obviously the best solution for the Kat indexer. The kfile plugin system, courtesy of KDE, comes in handy. kfile includes plugins for the extraction of metadata and plugins for the production of thumbnails. Each plugin is tailored to extract information from a predefined MIME type. Fortunately KDE ships with a lot of these plugins, so we had no need to build our own plugins for metadata and thumbnails.

Table 1 shows a list of the formats from which Kat is able to extract metadata.

What was lacking at the time we started developing Kat was a series of plugins capable of extracting the full text from documents. We decided to mimic the behavior of the other plugins by creating another class of kfile plugins called the "fulltext plugins." Then we started implementing plugins for the most popular formats like PDF and Word.

Table 1: Formats with Kat-readable Metadata

au	avi	bmp	cpp	dds
deb	diff	dvi	exr	flac
font	iso	ics	jpeg	m3u
mp3	mpc	ogg	palm	pcs
pdf	png	pnm	po	ps
rgb	rpm	rproject	sid	tga
theora	tiff	torrent	trash	ts
vcf	wav	xbm	xpm	

At the beginning, we thought it was a good thing to code plugins directly, but this soon turned out to be an impossible task. Some of the open source libraries created to allow the extraction of text from the various formats aren't written in C++, and some are not even written in C. Porting all the code into the project would have been a huge task. Another alternative would have been to link our code to the available libraries, but this also turned out to be impractical: while it is easy to find standalone utilities for extracting text from files, it is not so easy to find libraries.

The solution was to let our code call external tools, called helpers, for the extraction of the text from files. Although this technique is not so practical from a user's point of view, it allowed us to create a lot of plugins and to give Kat the possibility of handling many common desktop fotmats, such as HTML, RTF, PDF, PS, DocBook, TEX, DVI, and the native formats of the MS Office, OpenOffice, and KOffice suites.

In the future, when libraries are available, we will be extremely happy to link

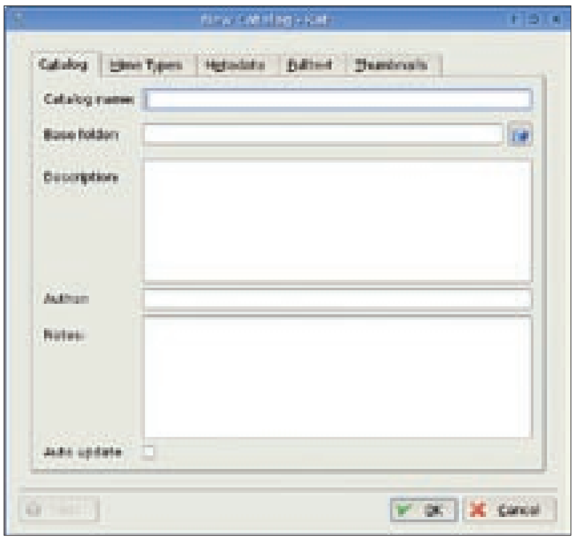


Figure 2: Creating a new catalog.

to them instead of having our users install a long list of additional programs.

Encoding and Recoding

Once Kat receives the text from one of its helpers, it must:

- Guess the language in which the document is written.
- Guess the encoding used in the text.
- Decode the text using the right decoder.
- Recode the text in a more practical encoding (UTF8).

If you have ever tried opening a file in a format that is different from the format in which it was saved, you know that even a small change in the character encoding can cause errors and illegible text. These errors are compounded if the text includes a variety of languages and alphabets. Kat must therefore correctly read the text in its original format and save it to a single known format for uniform processing.

The Unicode format was developed as a single encoding for all languages and alphabets. Unicode is a single huge table of more than 100,000 characters. Unfortunately, the Unicode format requires 4 bytes for each character, which means the resulting document will require 4 times the space required by an ASCII file. To address this issue, two abbreviated versions of Unicode have been developed: UTF8 and UTF16, which require only one byte or two bytes, respectively. To save space, Kat uses UTF8.

To correctly interpret the original text, Kat has to guess the language and encoding of each document. This problem is solved through computational linguistics. Each language or encoding has certain characteristics that make it easy to guess.

The algorithm used for guessing the language or encoding is called ngram-based document categorization. This algorithm divides the text into small pieces, each of which has a length between one and five characters. The various pieces, which are called ngrams, are then sorted by the number of times they occur in the text. This sorted list of ngrams is called a linguistic profile. The linguistic profile can serve as the fingerprint for a specific language. If two documents are written in the same language, no matter what domain they belong to, they will present the same linguistic profile.

In collaboration with the author of a library for language guessing, we devel-

Caching

One word about caching. Many users complain about the size of Kat's database. Kat caches at least one copy of anything that gets indexed. Caching allows you to preview a document without actually having to open it. Moreover, in the case of removable media, it allows you to access a copy of the document even if the media containing the original document is not present.

In order to reduce the amount of space required for the cache, Kat stores the text in compressed form. In a future version of Kat, users will be able to choose if they want a cache or if they want more space on their disks.

The users who choose to maintain a cache will also be able to display results like in Google, that is, with a couple of rows in which the found words are highlighted in the text of the document.

The caching feature also allows you to maintain a document history. For example, suppose you accidentally modified a document incorrectly. If you needed to get the original data back, you could open a cached copy from a date prior to the modification. This feature has not yet been implemented in Kat, but it is on our todo list.

The Graphical User Interface

Until now, the Kat project has focused on building a solid desktop search framework. This emphasis on the engine means that we have had comparatively less time to spend on producing a nice GUI. As soon as the work on the framework reaches completion, we will certainly dedicate much more time to the GUI.

You can think of the current Kat client as a prototype with the goal of demonstrating the usage of the API. It allows users to initiate searches, browse catalogs, and open documents. It also lets users search inside catalogs, but the *real* GUI, which is currently in development, will not even resemble the current version. We are drawing sketches and building mockups in order to perform usability tests. You can check the status of our GUI redesign in the pages of our Wiki.

Users of the new Kat GUI may not even notice they are using Kat. The search box will melt into the operating environment in such a way that it will simply be ubiquitous. Every time a user needs to search something, Kat will be there. As an example, we are working on adding Kat search capabilities to the standard file open dialog.

oped a method for extending the ngram technique to include encoding. The resulting library is now used by Kat for the first phase of language management.

The Qt library, which is part of the background of KDE, handles the task of decoding and recoding the text. The Qt library, which uses UTF8 internally, is able to decode text written in over 40 character encoding formats, including the ISO8859 series as well as many other European, Asian, and Middle Eastern encodings.

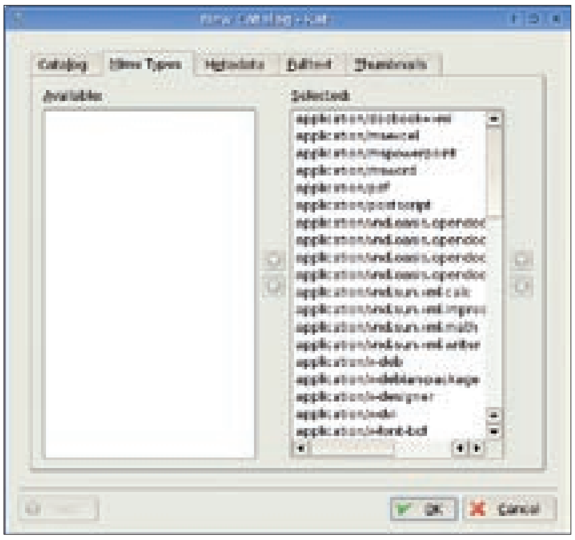
Reverse Indexing

Once the text is recoded into a normalized form, Kat must build what is called a "*reverse index*". A reverse index is a list in which each word of the document is linked to the documents in which the word appears. The normal data flow is document --> words. When we access the data in a reverse index, the structure is opposite: word --> document. Creating a reverse index may seem a trivial task, but it isn't. A reverse index involves concepts such as:

- sentence tokenization (the division of a sentence into words);
- word stemming (the ability to provide a match for single/plural words or conjugated forms);
- stop-word management (the exclusion from the index of frequent words that are not useful for searching, such as articles, pronouns, and so on);
- scoring by proximity and rank.

The algorithms used in the first releases of Kat proved too slow and inefficient, so for the next release, we are planning to replace these algorithms with external libraries specializing in full text storage and retrieval. One of the possible choices is Lucene, a library from the Apache project.

Lucene was originally written in Java and is available in various ports, like the C# port used by Beagle and the C++ port we are planning to use. In order to simplify the process of adopting Lucene, we have asked the author of Lucene to port it to C++/Qt, the environment used in KDE. He accepted our proposal and has created the CLuceneQt library. In the



application/docbook+xml
application/msexcel
application/mspowerpoint
application/msword
application/pdf
application/postscript
application/vnd.oasis.opendoc
application/vnd.oasis.opendoc
application/vnd.oasis.opendoc
application/vnd.oasis.opendoc
application/vnd.sun.xml.calc
application/vnd.sun.xml.impress
application/vnd.sun.xml.math
application/vnd.sun.xml.writer
application/x-deb
application/x-debianpackage
application/x-designer
application/x-dvi
application/x-font-bdf

Figure 3: The Mime Types tab manages the Mime types available to Kat.

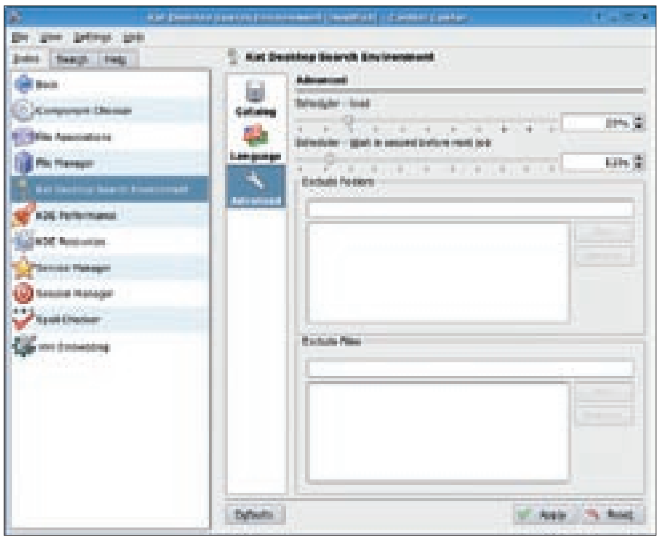


Figure 4: You can access advanced configuration options through Kat's Control Center module.

following months, we will collaborate with him to introduce this library into the Kat framework.

Another option for indexing is the FT3 full text indexer. The FT3 indexer is an implementation of the reverse indexing techniques featured by Lucene. FT3 uses SQLite3 for storage, the same database Kat uses. The fact that Kat is based on the same database back-end as FT3 could make the introduction of FT3 algorithms into Kat extremely simple. Our ultimate goal is to give users the possibility of choosing among different back-ends.

The Daemon

One important feature of Kat is its ability to automatically update the index when a change occurs in one of the files or directories that have been previously indexed. This update relies on the signals emitted by the inotify kernel module. Every time a file is created, deleted, moved, or renamed, inotify emits a signal that every application can intercept and act upon.

It is important to note that inotify is a pretty recent addition to the Linux kernel. The inotify kernel module has become an official part of the Linux kernel tree only in version 2.6.13. This means that, if you want to benefit from Kat's ability to automatically update the index, your Linux system must be based on a recent version of the Linux kernel. Patches for older kernel versions are available, but we definitely suggest that users update their Linux systems. Updating your system is an easy task today, because the new versions of all major distributions (Mandriva, Ubuntu, Suse, and other distros) ship with an inotify-enabled kernel.

In order to receive the notifications from inotify, Kat needs to cooperate with a daemon service that runs in the background. The Kat daemon receives the signals from inotify and updates the index. Moreover, every time the system is shut down and rebooted, the Kat daemon checks if the state of the file system has remained the same.

The daemon is the most delicate part of Kat, since it has to run without interruption throughout all user sessions. The workload of the daemon is managed by a scheduler, which can be configured by the user.

The actual implementation of the scheduler is based on time slices. If you configure Kat to use 10% of the CPU time, and if an indexing job takes 10 minutes, Kat will remain idle for 90 of the next 100 minutes. We are aware of the limitations of this solution, and we are planning to develop a new version of the scheduler that will be activated whenever the computer becomes

Context Links

Some months ago, we had the opportunity to discuss our goals and strategies with the members of the team developing the Tenor context link engine. Tenor's main goal is to produce a tool that will trace the links that connect different pieces of information stored in a computer.

Scott Wheeler, principal author of the Tenor concept, had this to say about his project: "The cool thing with Tenor is that it's asking bigger questions than the recent round of desktop search tools. It goes beyond 'Which of my files contains this information?' though it does do that too. But the point is, it also says, 'What's related to this?' and is able to search through those relationships."

The idea is to keep track of the contextual information we are throwing away now. If a user saves an image attached to an e-mail message, a lot of relevant context is lost forever: Who sent me the picture? Does my address book contain the address of the sender? Did I receive other messages from the sender? Did he send more graphic files in his other messages? Where else have I used this picture? A traditional desktop search engine would not be able to answer these simple questions.

In order to be effective, this kind of information has to be collected by the system in the most automated and transparent way. The users should not even notice that such a process is running in the background. And, most important, the process will require the active collaboration of the applications. Every time a user creates, deletes, or moves a piece of information, the application used for that action should notify Kat so that the information will not be lost.

A huge part of this context information comes from documents stored in the file system, so Kat is the perfect base for a contextual linkage engine like Tenor. We are currently reworking our API to make it easier to add a contextual layer to our storage schema. When we have finished, we will begin integrating context linkage into Kat.

Supporting Kat

We are planning to test Kat in a large set of environments using a vast variety of documents. This plan requires time but also equipment, and equipment means money. Kat is an independent project that is not sponsored by a large company, which means we must depend on the community for support. We are currently searching for a sponsor. Mandriva has kindly offered to host our web site, but we still need sponsors for other aspects of the project.

Money is not the only thing we need. If you have unused equipment or computer parts, please contact us. Setting up a testing facility would definitely make the difference and shorten development time. In particular, we would appreciate donations of not-so-old notebooks (>1.5 Ghz), extra large hard disks (>100 Gb), memory modules (for both notebooks and desktops), and so on. See the Kat website for more on how to make a donation.

idle and deactivated when the computer resumes activity.

The Application Programming Interface

One of the principal goals of the Kat project is to deliver a complete application framework for desktop searches so other developers can add search capabilities to their applications. We are working now to build comprehensive documentation of every class and every method of the Kat API.

The Kat web page contains a link to the API documentation. The documentation uses Apidox, an automated documentation tool that reads the special comments inserted in the code and builds nicely paginated documents that resemble the documents of the Qt library.

Using Kat

Kat is starting to find its way into more mainstream Linux distros. If your Linux distribution doesn't include Kat, you can obtain it from the Kat website [1]. The website includes source code as well as packages for Suse, Mandriva, Fedora, and Debian.

Kat Version 0.6.4 (the version shipped with Mandriva 2006) and later versions create a home catalog automatically during installation, so the average user will never need to cope with catalog management.

When you run Kat, a search dialog asks you to insert the word you want Kat to search for inside the catalog (Figure 1). The searches allowed in the current version of Kat are rather simple: one single word at a time. We are currently working on advanced search techniques that will allow complex queries with boolean operators (AND, OR, NOT, and so on). After you enter the word, press the Search button and the results appear in the bottom part of the window.

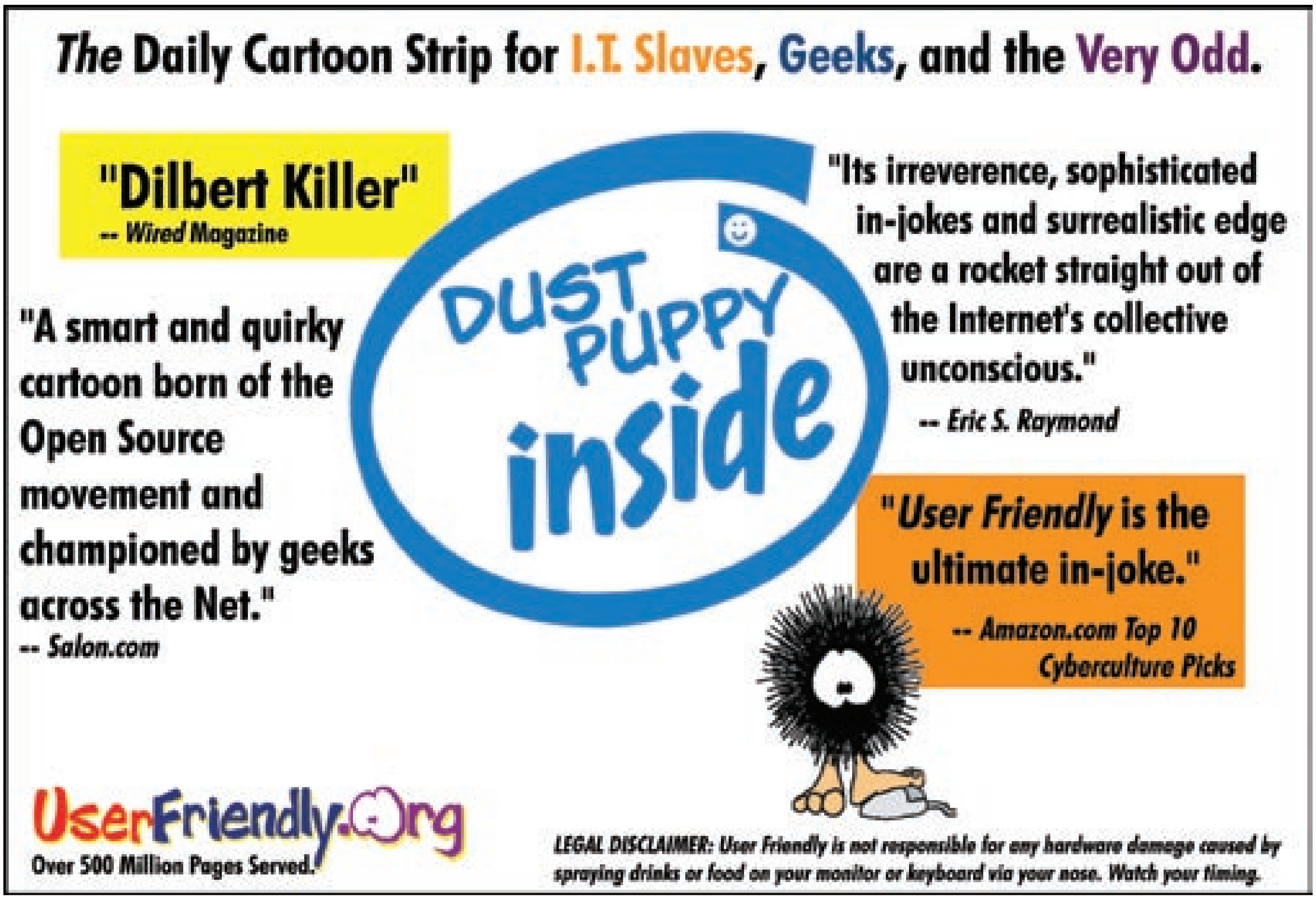
If you need to create a new catalog for indexing another folder, select *Catalog | New*. A dialog box requests information on the catalog you wish to create, such as the name, description, notes, author, and, last but not least, the option for auto-updating the catalog (Figure 2). By enabling this auto-update option, the user instructs the daemon to watch the folder in order to react to changes, such as file creations, deletions, or modifications. If the catalog is built for a read-only medium like a CD-ROM, it makes no sense to enable the auto-update feature, because the files will never need to be reindexed.

There are many options for configuring the index procedure. The other tabs visible in the window, Mime Types (Figure 3), Metadata, Fulltext, and Thumbnails, allow the user to select which types of files will be indexed and which will simply be ignored.

As soon as the user clicks on the OK button, the daemon starts building the index for the newly created catalog. You can monitor the status of the indexing process by clicking on the Kat icons in the kicker. ■

INFO

[1] Kat website: *http://kat.mandriva.com/*


The Daily Cartoon Strip for I.T. Slaves, Geeks, and the Very Odd.
"Dilbert Killer" -- Wired Magazine
"A smart and quirky cartoon born of the Open Source movement and championed by geeks across the Net." -- Salon.com
DUST PUPPY inside
"Its irreverence, sophisticated in-jokes and surrealistic edge are a rocket straight out of the Internet's collective unconscious." -- Eric S. Raymond
"User Friendly is the ultimate in-joke." -- Amazon.com Top 10 Cyberculture Picks
UserFriendly.Org
Over 500 Million Pages Served.
LEGAL DISCLAIMER: User Friendly is not responsible for any hardware damage caused by spraying drinks or food on your monitor or keyboard via your nose. Watch your timing.