

LISFS: a Logical Information System as a File System

Yoann Padioleau
École des Mines de Nantes
4, rue Alfred Kastler
Nantes, France

Yoann.Padioleau@emn.fr

Benjamin Sigonneau
IFSIC/IRISA
Campus de Beaulieu
Rennes, France

benjamin.sigonneau@irisa.fr

Olivier Ridoux
IFSIC/IRISA
Campus de Beaulieu
Rennes, France

ridoux@irisa.fr

ABSTRACT

We present Logical Information Systems (LIS). A LIS can be viewed as a schema-less database whose objects are described by logical formulas. Objects are automatically organized according to their logical description, and logical formulas can be used for representing both queries and navigation links. The key feature of a LIS is that it answers a query with a set of navigation links expressed in the same logic as the query. As navigation links are dynamically computed from any query, and can be used as query increments, it follows that querying and navigation steps can be combined in any order.

We then present LISFS, a file-system implementation of a LIS, where objects are files or parts of files. This has the benefit to make LIS features available right now to existing applications. This implementation can easily be extended and specialized through a plug-in mechanism.

Finally, we present some applications in the field of personal databases (e.g., music, images, emails) and in the field of software engineering.

Categories and Subject Descriptors

E.5 [Files]: Organization/structure; E.5 [Files]: Sorting/searching

Keywords

File system, logic, querying and navigation

1. LOGICAL INFORMATION SYSTEM

The framework of *Logical Information Systems* (LIS) manages data in a generic and flexible way [6]. This framework offers an alternative to hierarchical systems (e.g., UNIX file-system), boolean systems (e.g., Google) and relational systems (e.g., relational databases). It relies on a variant of *formal concept analysis* [7, 5].

The storage structure, called *formal context*, holds objects that are described by logical formulas. A context can be queried or seen as a navigation medium at any time. *Interrogation* answers to

a query, expressed as a logical formula, by computing the set of objects whose description entails the query. *Navigation* introduces the notion of a *place*, also expressed as a logical formula, and computes both the set of places accessible from the current place and the set of objects located in the current place. This amounts to computing the difference between a formula (the current place) and the formulas describing accessible places. If such differences are carefully computed, they form the basis of a progressive navigation tool.

In a LIS, an object can have two kinds of properties. *Intrinsic* properties are computed from the content of the object, whereas *extrinsic* properties are given by the user on the basis of criteria that cannot be computed from the object.

A LIS behaves like a schema-less database, the organization structure being computed dynamically from the logical descriptions of the objects.

In a database system as in a LIS, queries are *intentional*. The most striking difference between a database system and a LIS is the nature of the answer. Whereas it is *extensional* in a database system, i.e. objects, in a LIS the answer to a query is also intentional, i.e. formulas. Therefore, the search can be progressive through an iteration process: ask for a query, pick up an answer to complement the query, etc. This is analogous to a dialog between a customer (C) and a shop assistant (SA):

C: I want to buy flowers! What do you have?

SA: Do you have any idea of the color, kind of flower or size of bouquet?

C: I want a big bouquet! What color do you have?

SA: Red, white or yellow.

...

2. LOGIC FILE SYSTEM

LISFS (Logical Information System File System) implements the principles of a LIS at the system-level under the interface of a file system [14, 15]. Objects are files or parts of files, and places are presented as directories. Such a system approach makes logical access to files a commonplace, regardless of their types and whatever application is used [4, 14].

Modern OSes consider a file system as a set of methods that allows to manage files and directories. Under Linux, VFS (Virtual File System) acts as an interface of which every method is virtual. A file system is just an implementation of those methods; this is the case of LISFS, which runs over the Linux kernel. As shown in Figure 1, LISFS has a *plug-in* mechanism. There are two kinds of plug-ins: *logics* are used to describe, query and classify objects; *transducers* compute intrinsic properties of files.

With LISFS, paths are seen as formulas. LISFS embeds boolean querying: / reads “and”, | reads “or” and ! reads “not”. For

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE '06 Shanghai, China

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

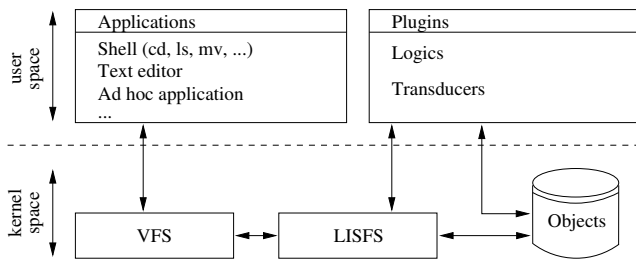


Figure 1: LISFS architecture

instance, the query `cd /music/!bad/disco|rock` would select disco or rock music files that are not tagged as bad.

It should be emphasized that LISFS operates on two levels. At the first level, called *inter-file*, objects are files. At the second level, called *intra-file*, objects are parts of files [15].

By design, querying and navigation within LISFS must be compatible with an interactive use. Experiments with the current implementation show that it is the case up to an order of 100,000 objects. While the stress is put on a fast answering to queries, indexing might take longer. However, only the initial indexing of large data might take long. Once it is done, the index is stored persistently on disk. If the data changes, re-indexing is incremental and faster.

3. APPLICATION EXAMPLES

3.1 Managing a music database

Music files are easily managed within LISFS. As a matter of fact, music files generally come with meta-data; e.g., MP3 files embed the name of the artist, the title of the song and so forth. Such meta-data can be extracted by a transducer and used to describe a music file.

Using a traditional UNIX shell as an interface, the `cd` command queries the database and the `ls` command lists the navigation links, as shown in the example below:

```
[1] cd /lfs/music/year:[1980..1990]
[2] cd !genre:Comedy
[3] cd time:<7min
[4] cd .ext
[5] playmp3 *
...
[6] cd /lfs/music/genre:Disco/
[7] ls
artist:BeeGees/ artist:DonnaSummer/
[...]
year:1976/ year:1977/
[...]
```

Command 1 selects music files from the '80s (using an interval logic for years), command 2 excludes comedy from the selected files (! denotes negation) and command 3 selects only short songs (using a time logic). From then, the user can switch to the extension of its query (command 4) and listen to it (command 5). Command 6 selects disco music and command 7 then asks for navigation links. It can be seen that they can be used as legitimate queries so as to refine the current query.

Those who prefer to use a graphical interface, as can be seen in popular tools such as iTunes, may find the use of command-line in the above example awkward. This is not really a problem: since LISFS is implemented as a real file system, every application can benefit from it. Therefore, a music manager *à la* iTunes is simply

a specialized interface over LISFS; see Figure 6 for a screenshot. This approach has two advantages. First, the code of the interface is very simple, since the hard work of indexing, querying and answering relies entirely on LISFS. Second, the resulting application is more powerful than most popular tools. For instance, iTunes allows the user to browse a music library by genre then by artist but not the contrary; this is no problem with our application.

3.2 Managing a bibliography database

The intra-file mode of LISFS can be used to manage a bibliography database stored in a BibTeX file. In this case, the objects are the entries of the BibTeX file, and a transducer extracts properties such as author, title, etc. LISFS was successfully used to manage a 100,000 line BibTeX file.

Navigation inside a BibTeX file is close to some data-mining work. For example, the navigation links resulting from a query can easily be used to look for frequent co-authors of a given author.

Moreover, after performing some query, the user is presented a partial view of the original file, showing only the relevant entries. This view can not only be further queried, but also edited. Changes are propagated to the original file in a consistent way, and the view is consequently updated.

3.3 Managing e-mails

E-mails can also be managed within LISFS. Experiments were achieved on repositories holding up to 160,000 messages. A transducer extracts author, subject, thread, date, etc., from a message and this information is used to index e-mails.

By assigning extrinsic properties to the e-mails, the user can organize its messages with some kind of virtual folders. It should be noticed this allows e-mails to naturally belong to several virtual folders.

In such a context, a user is likely to be looking for an old e-mail. Therefore, full-text indexing of messages seems to be a good idea. However, doing it naively in LISFS would lack efficiency, since there would be too many navigation links available. To alleviate this problem, Glimpse[12] is used for full-text indexing of the messages, and the file system interacts with Glimpse to process some queries. Queries of the form `cd contains:foo` call Glimpse and select files that contain the text "foo"; however, rather than showing these files, navigation links that lead to them are computed. Please note that the integration of external tools in the query/navigation process is not part of the LIS theory, but comes as a practical extension of LISFS.

3.4 Homedir

Apart from the few examples described above, LISFS is used to managed many other kinds of files such as source code of programs, man pages, LaTeX files, etc. From then, the idea of managing a user's entire home directory, which is nothing more than a collection of numerous and heterogeneous files, sounds appealing. As we saw in previous examples, the user benefits from the different applications of LISFS on various kinds of files.

Moreover, LISFS allows him to perform *transversal* queries on different kinds of data. For example, let us consider a user looking for files that contain the word "synchronization" but who does not remember exactly their types nor their places. With a home directory managed under LISFS, he will be able to use the query `cd contains:synchronization` to select every data that mention the word "synchronization", regardless whether it is in an e-mail, a documentation, a program, etc. Then, the system itself will propose navigation links for the user to complement the query.

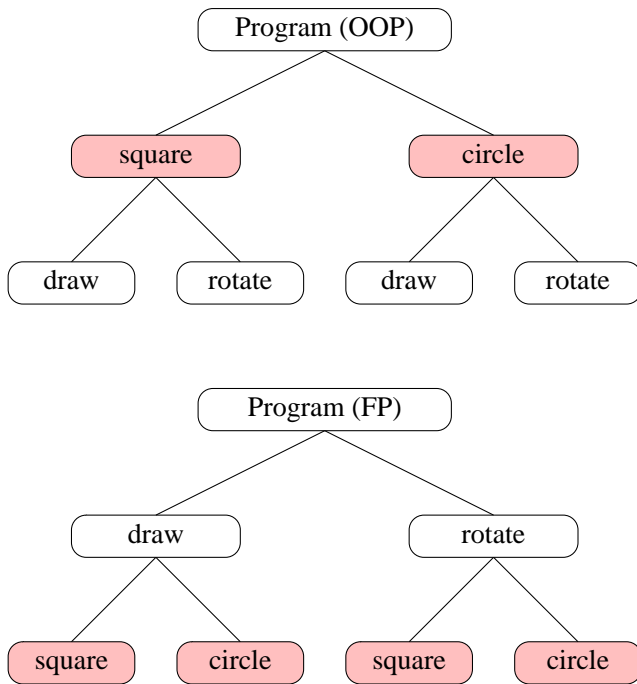


Figure 2: File contents are rigid

4. SOFTWARE ENGINEERING EXAMPLES

4.1 Linux Kernel Source

LISFS has been used to manage the Linux kernel source tree. In this case, intrinsic properties of the files are the names of the function they contain (as extracted with the `ctags` command). The files also contain extrinsic properties that corresponds to their path in the original source tree (e.g., `driver`, `fs`, `include`, etc.).

With such a huge source tree, one would like to classify them according to its current needs: by file type (includes, C files, text documentation, ...), by architecture (x86, PPC, ...), by module (drivers, virtual memory manager, ...) and so on. However, the organization of the Linux kernel source tree suffers from the limitations of hierarchical file systems and therefore forces the developer into using a fixed organization scheme (currently, by type, then by module, then by architecture). Some directories therefore appear in different places (e.g. there are 4 sound directories), the organization is not always coherent and is hard to understand.

One answer to these problems is to use specific tools, such as LXR [1], a cross-referencing tool for relatively large code repositories.

Another solution is to use LISFS, with which these problems become irrelevant. The classification scheme is not fixed anymore, and the user can select the set of files dealing with memory handling, should they be includes, C files or documentation. Or he could have retrieved every include files with the same ease.

4.2 Managing source code: intra-file level

The intra-file mode of LISFS is well adapted to managing source code. As a matter of fact, the content of a source code file is generally organized hierarchically. Thus, it shows up the same rigidity as traditional hierarchies. This has been recognized as the “tyranny of the dominant decomposition” [17].

Such an example is shown on Figure 2, which presents a simple program that deals with primitive graphics operations. Two versions of the program are presented. In the first version, written in an object-oriented language, the stress is first put on the nature of the data (square or circle) and then on the operation to perform on that data (draw or rotate). On the contrary, the second version, written in a functional programming language, first insists on the operation to perform, then on the data on which the operation applies.

Therefore, it will be easy to focus on a particular data type in the first program ; in the second program, this will be hard to do since the information about a particular data type is scattered within the different functions. If the user want to focus on a particular function, the same problem arise: it will be easy to do in the FP version, but quite hard and error-prone in the OOP version.

LISFS intra-file mode alleviates this problem. When entering a source code file, LISFS will show the user that he can get a partial view of the file dealing with square, circle, draw or rotate. Whatever programming paradigm is used, the user will thus be able to focus on its current needs.

Furthermore, a source code contains other information, such as debugging statements, comments, assertions, etc. As recognized by Aspect-Oriented Programming [11], this cross-cutting information is generally not handled in an easy way for the programmer because they are scattered in the program. With LISFS intra-file mode, it becomes easy to manage such cross-cutting concerns: e.g., the user may ask for a partial view of his file with comments and without debugging statements.

4.3 Software Component Retrieval

Software reuse requires that programmers be able to locate reusable components in software repositories. We applied LISFS to querying in software repositories [16].

As an illustration, we indexed methods of a Java package, using three different indexing schemes: a formal scheme, a semi-formal one, and an informal one. The formal one captures object-orientation by combining type isomorphism axioms and inheritance relations. The semi-formal scheme captures naming conventions and the informal one captures keywords in comments.

By combining those three orthogonal indexing schemes in LISFS, the user is able to find Java methods easily. Indeed, the different indexing schemes are intertwined and can be used seamlessly: the user can browse the package by class (as in the Javadoc documentation tool), by input or output type (*i.e.* look for a method that take a Window and returns string) or can query the repository by extracted keywords.

5. RELATED WORKS

5.1 Integrated Development Environments

Most ideas presented in LISFS have already been in use in several IDEs or advanced editors. For instance, the idea of querying a source code has already been developed in JQuery [13]. Another example is the notion of partial views of a file, which is also present in Emacs. However, we are not aware of an IDE that would have *all* the features LISFS provides to a programmer.

Moreover, we believe LISFS is a more general solution. Indeed, most IDEs are devoted to a particular language or paradigm, while LISFS can be used in conjunction with any of these through its plugin mechanism. What’s more, an IDE generally enforce the user to using a particular set of tools (editor, code browser, etc.). LISFS being implemented at low-level, its services can be reused through any already existing application. The user can therefore use the tools he is already accustomed to.

In fact, LISFS follows UNIX philosophy in that it allows the programmer to combine the tools he wants to use to build its own developing environment.

5.2 Advanced File Systems

Non hierarchical file systems were already proposed, e.g. SFS [9], CATFS [8], Nebula [2], HAC [10]. Each of them brings some interesting features like automatic assignment of intrinsic properties, directories seen as queries, powerful querying or the notion of view. Nevertheless, none of them achieved to have all those features at once, nor did they provide such a seamless integration of querying and navigation as LISFS does.

Moreover, they do not provide an intra-file level. On the contrary, LISFS successfully generalized the above principles to the contents of files.

5.3 Personal Databases

LISFS can be seen as an answer to the problem of personal document retrieval. This concern has recently been addressed by the computer industry in two different ways. On the one hand, tools like Google Desktop indexes documents in a database; on the other hand, file system approaches, as done in Apple's Spotlight, store indexes as file system attributes.

Both those approaches provide a searching tools in the form of a user application. LISFS goes further in the file system way in that not only the indexes are stored in the file system, but also the file system *is* the searching tool.

Moreover, LISFS is a more versatile tools. It is easy to write plugins that make it aware of new file types, source code of different languages being an example.

6. CONCLUSION

We presented Logical Information Systems that allow a collection of objects to be organized within a schema that is neither defined *a priori* nor hierarchical. The collection can be browsed by combining querying and navigation freely. Managing the collection is easy: intrinsic properties are automatically updated by transducers when a file content is changed, and extrinsic properties can always be updated by the user. Therefore, the collection is automatically and incrementally re-organized.

LIS provides a highly expressive framework to navigate, query, update and organize data. A file system implementation allows every application to take advantage from it.

Future releases will provide relations between objects and data-mining operations. This will allow us to build debugging tool on top of LISFS [3].

We believe that software engineering is a good field of application for LISFS. Software engineering deals with objects of different natures: programs, specifications, data (e.g. for testing purpose), that are linked through complex relations. They organize in versions and configurations, can be described in UML diagrams, can be decomposed in views, facets, aspects...

Those organizations are often considered separately. We believe they should be considered as alternative views of the programmer's information system. LISFS can offer this service to the programmer by exploiting heterogeneous data and extracting views corresponding to a particular situation: component retrieval, fault localization, conformity checking w.r.t specifications, and so on.

More information about Logical Information Systems can be found on <http://lfs.irisa.fr>. In particular, LISFS can be downloaded from <http://lfs.irisa.fr/~pad/LFSWEB/>.

7. ADDITIONAL AUTHORS

- Sébastien Ferré (IFSIC/IRISA, ferre@irisa.fr)
- Mireille Ducassé (INSA/IRISA, ducasse@irisa.fr)
- Olivier Bedel (IFSIC/IRISA, olivier.bedel@irisa.fr)
- Peggy Cellier (IFSIC/IRISA, cellier@irisa.fr).

8. REFERENCES

- [1] Linux cross-reference project. Available on <http://lxr.linux.no/>.
- [2] C. M. Bowman, C. Dharap, M. Baruah, B. Camargo, and S. Potti. A File System for Information Management. In *Int. Conf. Intelligent Information Management Systems (ISMM)*, 1994.
- [3] T. Denmat, M. Ducassé, and O. Ridoux. Data mining and cross-checking of execution traces. A re-interpretation of Jones, Harrold and Stasko test information visualization. In T. Ellman and A. Zisman, editors, *20th Int. Conf. on Automated Software Engineering*. ACM Press, Nov. 2005.
- [4] S. Ferré and O. Ridoux. A file system based on concept analysis. In Y. Savig, editor, *Int. Conf. on Rules and Objects in Databases*, LNCS 1861, pages 1033–1047. Springer, 2000.
- [5] S. Ferré and O. Ridoux. A logical generalization of formal concept analysis. In G. Mineau and B. Ganter, editors, *Int. Conf. on Conceptual Structures*, LNCS 1867, pages 371–384. Springer, 2000.
- [6] S. Ferré and O. Ridoux. Introduction to logical information systems. *Information Processing and Management*, 40(3):383–419, 2004.
- [7] B. Ganter and R. Wille. *Formal concept analysis — Mathematical Foundations*. Springer, 1999.
- [8] D. Giampaolo. CAT-FS, a Content Addressable, Typed File System. Master's thesis, Worcester Polytechnic Institute, 1993.
- [9] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. Jr. Semantic File Systems. In *ACM Symp. Operating Systems Principles*, 1991.
- [10] B. Gopal and U. Manber. Integrating Content-Based Access Mechanisms with Hierarchical File Systems. In *ACM Symp. Operating Systems Design and Implementation*, 1999.
- [11] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP '97*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, 1997.
- [12] U. Manber and S. Wu. GLIMPSE: A Tool to Search Through Entire File Systems. In *USENIX Winter Conf.*, 1994.
- [13] E. McCormick and K. De Volder. Jquery: finding your way through tangled code. In *OOPSLA Companion*, pages 9–10, 2004.
- [14] Y. Padioleau and O. Ridoux. A logic file system. In *USENIX Annual Technical Conference*, 2003.
- [15] Y. Padioleau and O. Ridoux. A parts-of-file file system. In *USENIX Annual Technical Conference*, 2005.
- [16] B. Sigonneau and O. Ridoux. Indexation multiple et automatisée de composants logiciels orientés objet. In J. Julliard, editor, *Approches Formelles dans l'Assistance au Développement de Logiciels*, pages 43–57, 2004.
- [17] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. Sutton Jr. *N* degrees of separation: Multi-dimensional separation of concerns. In *ICSE*, pages 107–119, 1999.

APPENDIX

A. DEMO SCENARIO

A.1 LISFS basics: managing a music library

The first part of the demo shows LISFS basics over a context of MP3 music files. Within a standard file explorer, we demonstrate the query and navigation process:

- how to query LISFS and how to get help from LISFS through navigation links
- use of boolean queries
- use of additional logics (interval logic on numeric properties).

We then show how to give extrinsic properties to objects and how to update the context.

A.2 A generic low-level implementation

On the same MP3 music files context, we demonstrate how having LISFS makes logical access to files a commonplace, *i.e.* how the low-level design of LISFS makes its technology available to every regular application. To this effect, we replay the search of music files within a regular UNIX shell and open the music file in a mainstream music player.

Then, we show the *ad hoc* music player (see Figure 6) and emphasize that it is very easy to write such an application. Only the interface has to be written, the hard work of the query process being embedded in LISFS.

A.3 Intra-file level: managing source code

In the previous part of the demo, only the inter-file level has been shown. We now show how query and navigation at the intra-file level allows to make partial views of a file. To this end, the source code of a program is used. This amounts to:

- entering a file (lines of the file become objects)
- querying the content of a file to get a partial view of it
- getting help from LISFS navigation links to form a query
- viewing a partial file
- using boolean queries to select non-trivial views of the file.

After this, editing and automatic re-indexing of the file is presented.

A.4 Homedir management

Last, we present how LISFS can be used to manage heterogeneous data in a home directory with a particular emphasis on transversal queries:

- using views to restrict the search space
- using transversal queries, acting regardless of the file type
- using glimpse integration to process full-text queries over documents in the homedir.

B. SCREENSHOTS

Figures 3, 4, 5 and 6 show various applications running on top of LISFS.

Figure 3 shows Emacs using the intra-file mode of LISFS to edit a BibTeX file. On top of the figure, a UNIX shell is run inside Emacs. By looking at the current working directory, shown in the prompt line, it can be seen that the current query is a view on BibTeX files restricted to the publications from 1968. The last

command, `view ridoux.bib`, opened such a partial view of the file `ridoux.bib` in an Emacs buffer, visible in the bottom-half of the figure. One can verify that all entries relate to publications in 1968. Missing parts of the original file are shown as dotted lines (e.g., `.....:1`).

Figure 4 shows Kate, an advanced text editor shipped with KDE, running on top of LISFS. The two windows present two different views of the same C++ program. This file describes two classes Circle and Rectangle, that both have draw methods (among others), as described before in Section 4.2.

The top window is a partial view of the C++ file showing only parts relevant to the Circle class. The bottom window shows another partial view of this file, with only parts dealing with Draw methods. The query in the URL bar at the very top of the screenshot is relevant to the bottom window.

Figure 5 shows the Mozilla web browser visiting the webpage of LISFS author (<http://lfs.irisa.fr/~pad/>). This page, served by a regular Apache web server, is managed under LISFS. Therefore, it is possible to express LISFS queries within Mozilla. The query expressed on this screenshot restricts the webpage to documents or pictures (`type:Document|type:Picture`) that are not text files (`!ext:text`).

Please note that you can reproduce this query in your favorite web browser.

Figure 6 shows a clone of Apple's iTunes music player. This clone was designed with LISFS in mind. Therefore, about 80% of its source code deals with the graphical interface, since the hard work of indexing and querying the music library relies on LISFS features. The user can browse its music library by genre, by artist or by album, in any order. He can also express more complex queries in the advanced query textbox.

```

drwxr-xr-x 1 pad pad 17 Jun 14 14:49 ref:nls
-rw-r--r-- 1 pad pad 672 Jun 15 10:03 ridoux.bib
drwxr-xr-x 1 pad pad 4 Jun 14 14:49 synchro
-rw-r--r-- 1 pad pad 638 Jun 15 10:03 thesis.bib
drwxr-xr-x 1 pad pad 7 Jun 14 14:49 title:Combinatory Logic
drwxr-xr-x 1 pad pad 9 Jun 14 14:49 title:Goto Statement Considered Harmful
drwxr-xr-x 1 pad pad 6 Jun 14 14:49 title:\'E}crits logiques
drwxr-xr-x 1 pad pad 17 Jun 14 14:49 title:{A} {R}esearch {C}enter for {A}ugmenting {H}uman {I}ntellec
drwxr-xr-x 1 pad pad 9 Jun 14 14:49 typeref:Article
drwxr-xr-x 1 pad pad 17 Jun 14 14:49 typeref:article
drwxr-xr-x 1 pad pad 13 Jun 14 14:49 typeref:book
~/lfs/symlinks/classicview/website/LFSWEB/demos/cd_parts_bibtex.query/year:1968 $ ls
author:D. C. Engelbart ref:curry:combinatory:1968 title:Goto Statement Considered Harmful
author:E.W. Dijkstra ref:dijkstra:goto:cacm:68 title:\'E}crits logiques
author:H.B. Curry ref:herbrand:ecrits:68 title:{A} {R}esearch {C}enter for {A}ugmenting {H}uman {I}ntellec
author:J.~Herbr ref:nls typeref:Article
author:R. Feys ridoux.bib typeref:article
author:W. Craig synchro typeref:book
author:W. K. English thesis.bib
domain: title:Combinatory Logic
~/lfs/symlinks/classicview/website/LFSWEB/demos/cd_parts_bibtex.query/year:1968 $ v ridoux.bib
-- *pad-sh3 (EShell ) Bot - L4001 - 10:05 0.02t -----
|.....:1
|@book{curry:combinatory:1968,
| author = "H.B. Curry and R. Feys and W. Craig",
| title = "Combinatory Logic",
| publisher = "North-Holland",
| comment = "second edition",
| year = "1968"
| }
|.....:2
|@Article{dijkstra:goto:cacm:68,
| author = "E.W. Dijkstra",
| title = "Goto Statement Considered Harmful",
| pages = "147--",
| journal = "Comm. ACM",
| volume = 11,
| year = 1968,
| keywords = "GOTO"
| }
|.....:3
|@book{herbrand:ecrits:68,
| author = "J.~Herbrand",
| title = "\'E}crits logiques",
| }
-- ridoux.bib (BibTeX Out1 ) Top - L1 - 10:05 0.02t -----

```

Figure 3: Using Emacs over LISFS

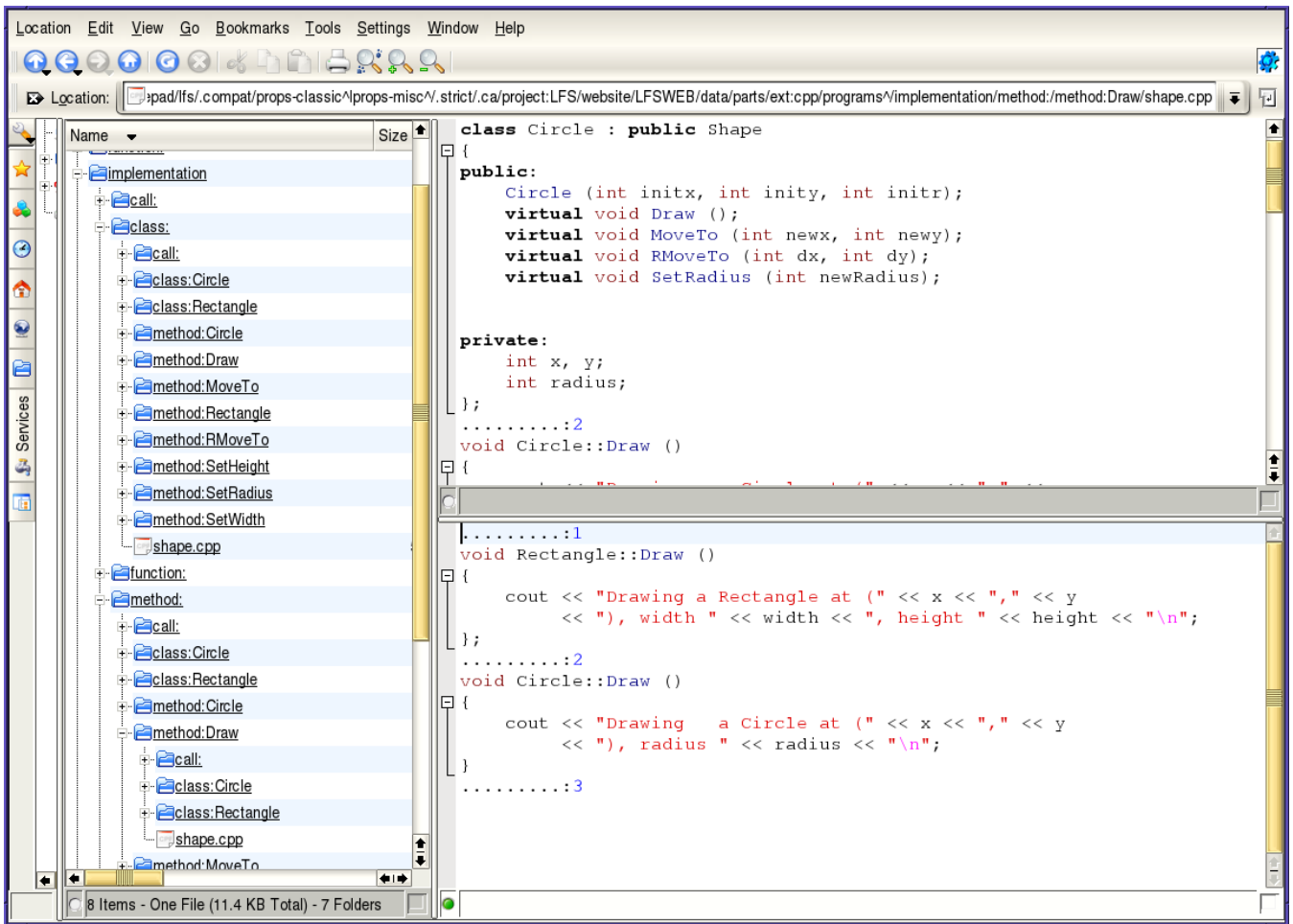


Figure 4: Using Kate (KDE text editor) over LISFS

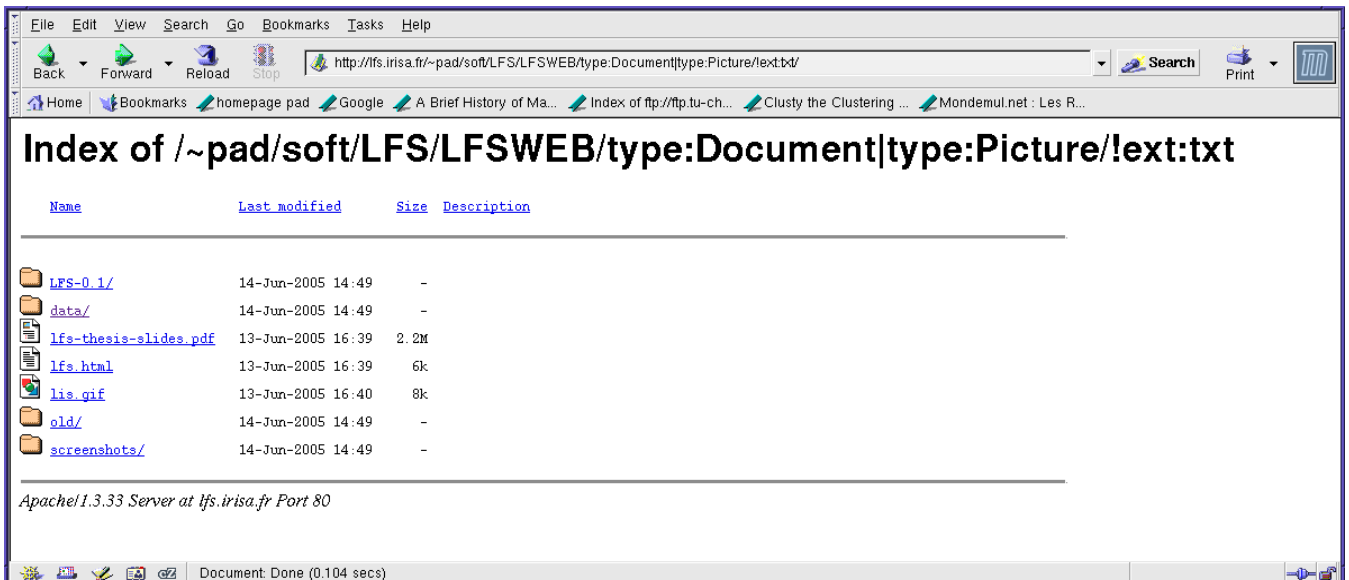


Figure 5: Using Mozilla over LISFS

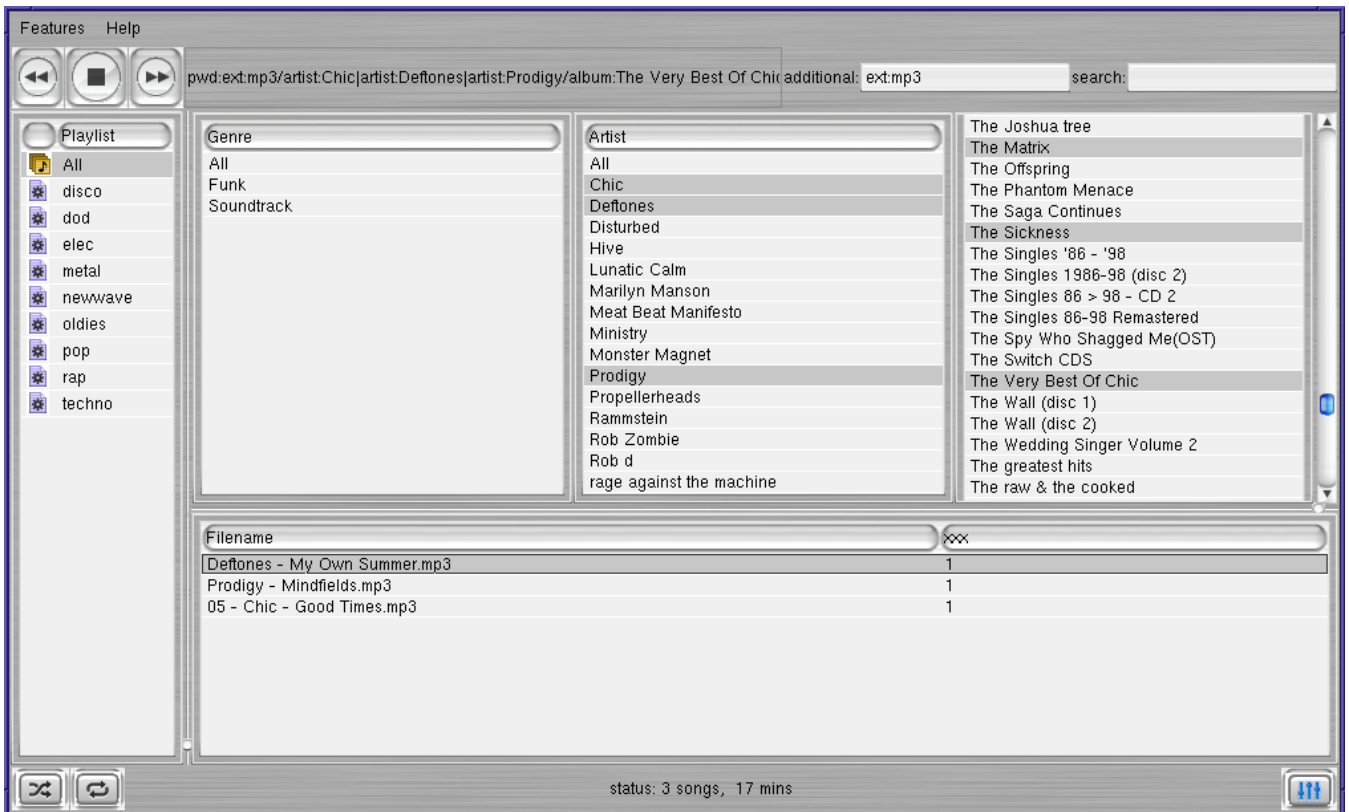


Figure 6: Graphical interface over LISFS for music database