

LISFS: a Logical Information System as a File System

Yoann Padioleau

Benjamin Sigonneau

Olivier Ridoux

Sébastien Ferré

IFSIC/IRISA

Campus Universitaire de Beaulieu

35 042 Rennes Cedex, France

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 demonstrate that building specialized interfaces for visualizing databases can be done easily through LISFS navigation.

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 [5]. 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, 4].

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.

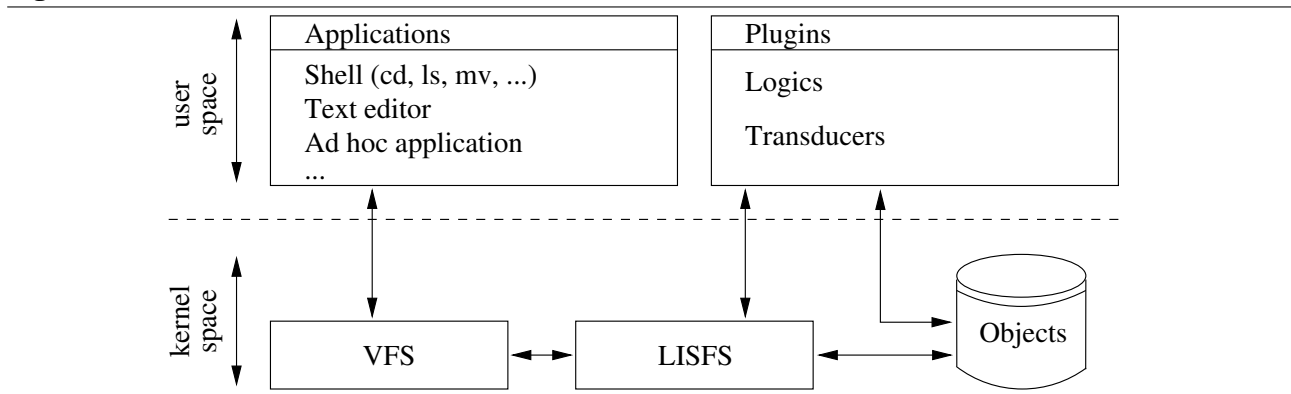
In a LIS, an object can have two kinds of properties. *Intrinsic* properties are computed from the content of the object, whereas *extrinsic* properties are given by the user on the basis of criteria that cannot be computed from the object.

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.
...

Figure 1 LISFS architecture



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 [13, 14]. 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 [3, 13].

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 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 [14].

By design, querying and navigation within LISFS must be compatible with an interactive use. Current experiments 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.

To achieve speed, LISFS uses inverted indexes to manage its meta-data. The indexes are hash tables, which are accessed with the Berkeley DB library [12]. The indexes are persistently stored as B-trees [2]. Moreover, LISFS takes advantage of the atomic transactions of the Berkeley DB library to offer journaling [15].

3 Application Examples

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.

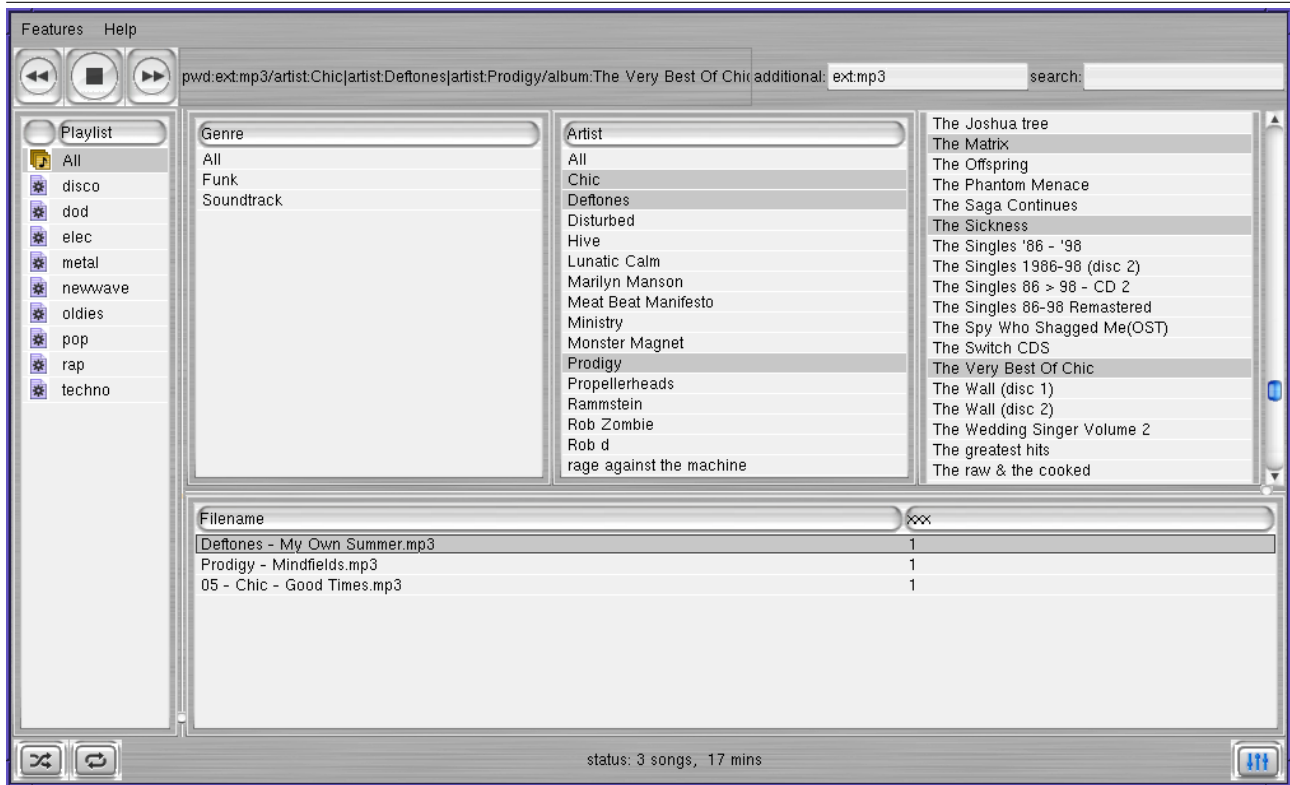
Using a traditional UNIX shell as an interface, the `cd` command queries the database and the `ls` command lists the navigation links, as shown in the example below:

```
[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

Figure 2 Graphical interface over LISFS for music database



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 *a la* iTunes comes for free: it is barely a specialized interface over LISFS. Such an interface was written; see Figure 2 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.

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

Navigation inside a Bib_TE_X 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.

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[11] 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.

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.

4 Demo scenario

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

- how to query LISFS and how to get help from LISFS through navigation links;

- use of boolean query;
- 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 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 standard file explorer and open the music file in regular music players.

Then, we show the *ad hoc* music player (see Figure 2) 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.

Intra-file level. 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, a bibliographic database stored as a Bib \TeX file is used. This amounts to:

- entering the file (lines of the file become objects);
- querying the content of the file to get a partial view of the file;
- 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. Next, we show how the intra-file level apply to files that does not hold a database, such as source code. For example, this allows the user to extract a table of content of an article from its \LaTeX source.

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.

5 Related Works

Relational Databases LISFS distinguishes from relational databases in several points. LISFS assets are a schema-less organization of objects and a seamless integration between querying and navigation. As navigation links computed by LISFS are expressed in the same language as queries, there is no need for the user to learn a query language. Moreover, object descriptions in LISFS are not limited to flat attributes; the use of specialized logics (e.g. date logic, string logic, interval logic, etc.) allows for rich descriptions.

We believe the low-level implementation as a file system is another point for LISFS. Being standalone applications, traditional databases are hard to interface with. On the contrary, LISFS is totally integrated in the operating system by design, thus LISFS technology is made available to every single application.

However, LISFS does not fully handle relations between objects. Managing arbitrary relations is still ongoing work [6].

Advanced File Systems Non hierarchical file systems were already proposed, e.g. SFS [9], CATFS [8], Nebula [1], 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.

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.

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.

References

- [1] 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.
- [2] Comer. D. The Ubiquitous B-Tree. *ACM Computing Surveys*, 1979.
- [3] Sébastien Ferré and Olivier Ridoux. A file system based on concept analysis. In Y. Savig, editor, *International Conference on Rules and Objects in Databases*, LNCS 1861, pages 1033–1047. Springer, 2000.
- [4] Sébastien Ferré and Olivier Ridoux. A logical generalization of formal concept analysis.

In G. Mineau and B. Ganter, editors, *International Conference on Conceptual Structures*, LNCS 1867, pages 371–384. Springer, 2000.

- [5] Sébastien Ferré and Olivier Ridoux. Introduction to logical information systems. *Information Processing and Management*, 40(3):383–419, 2004.
- [6] Sébastien Ferré, Olivier Ridoux, and Benjamin Sigonneau. Arbitrary relations in formal concept analysis and logical information systems. In F. Dau, M.-L. Mugnier, and G. Stumme., editors, *International Conference on Conceptual Structures*, LNAI 3596, pages 166–180. Springer-Verlag, 2005.
- [7] Bernhard Ganter and Rudolf 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’Toole Jr. Semantic File Systems. In *ACM Symp. Operating Systems Principles (SOSP)*, 1991.
- [10] B. Gopal and U. Manber. Integrating Content-Based Access Mechanisms with Hierarchical File Systems. In *ACM Symp. Operating Systems Design and Implementation (OSDI)*, 1999.
- [11] U. Manber and S. Wu. GLIMPSE: A Tool to Search Through Entire File Systems. In *USENIX Winter Conf.*, 1994.
- [12] M. A. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *FREENIX Track: USENIX Annual Technical Conf.*, 1999.
- [13] Yoann Padioleau and Olivier Ridoux. A logic file system. In *USENIX Annual Technical Conference*, 2003.
- [14] Yoann Padioleau and Olivier Ridoux. A parts-of-file file system. In *USENIX Annual Technical Conference*, 2005.
- [15] M. Rosenblum and J. K. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 1992.