

A Conceptual Model for Knowledge Management with reference implementation as a Filesystem

Shivam Sharma^{1,*}, Mengxiang Jiang¹ and Shraddha Sharma²

¹Texas A&M University, 400 Bizzell St, College Station, TX, US 77843

²Independent Researcher, 6 Petir Road, Singapore 678267

Abstract

We present a conceptual model for a knowledge management system (called *LIST*), its reference implementation as a Unix-based filesystem (called *LFW*), and utilization of this model for applications beyond file management, e.g. a git middleware (called *git_lfw*) which preserves file provenance automatically. In this paper, we describe how this conceptual model was born from a software project for unifying data sources (phase-1) to the current filesystem implementation (phase-2) and finally to a conceptual model specification (phase-3). The work advances conceptual modeling by providing a practical multi-model abstraction for knowledge representation, and in-line with ER-2026 theme on “Sustainability”, this work enables sustainability in two key ways: (1) sustainability of knowledge management for a human lifespan, and (2) environmental sustainability. More details about the project can be found at: <https://listspec.org>.

Keywords

Data Management, Knowledge Graphs, Domain-specific modeling

1. Introduction

There has been a desire among digital users and researchers alike to have a knowledge management system where knowledge units could be stashed without the *upfront* cognitive burden of thinking about “where to file” or “what to name”. An informal name was coined for such a system by [1, 2] – “*Laundry Basket Model*”. Here, the word *upfront* is key since users don’t want to avoid the organization of their knowledge, they only want to defer it and refine it when required. The current hierarchical filesystems are notoriously bad at this; renaming or moving files can break links and hence the organization.

To address this issue and achieve a laundry basket model, we present a conceptual model (or specification) for knowledge or document management called *LIST* – **Linked Immutably with Strong&Static Typing**, which can be implemented as a filesystem, through a database, or even as a document. While the acronym of *LIST* captures its two core axioms (discussed in Section 3), the name *list* itself encodes the feeling of a laundry basket model – where the document collection is presented to the user as “*just a list*”. We implement this specification as a filesystem called *LIST Filesystem Wrapper (LFW)*.

The paper proceeds as follows. We first describe in Section 2 the project phases in the way the project evolved. In Section 3, we briefly summarize the *LIST* specification, the *LFW* implementation, and then one of its applications beyond file organization. We conclude our work in Section 4.

2. Project Setup and Phases

In this section, we describe the three project phases, but first begin with a zeroth phase of literature survey and trying to understand the problem deeper. Somewhat surprisingly, the zeroth phase is our most significant contribution because it’s in this phase that we crawled through innumerable existing problems (reported by users and identified by researchers) to distill them down to the only two required solutions which finally became the two key axioms of our *LIST* model.

ER2026: Companion Proceedings of the 45th International Conference on Conceptual Modeling: Doctoral Consortium, Tutorials, Posters and Demos, October 5-8, 2026, St. John’s, NL, Canada.

*Corresponding author.

✉ shivam@tamu.edu (S. Sharma); mj294@cornell.edu (M. Jiang); shraddha004@e.ntu.edu.sg (S. Sharma)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2.1. Phase-0: Understanding the Problems

To gain insight into the problems and potential solutions, we did literature survey[3, 4, 5, 6, 7, 8, 9, 10, 11, 12], performed analysis of internet forum discussions on knowledge management¹, and utilized the feedback on our prior contributions[13] in knowledge management software, to derive a matrix of problems, desired features, and solutions. While the matrix has been discussed in detail on our project web-site (listspec.org), here we discuss the two identified problems in the next section — the problem of *fragmentation* and the problem of *longevity or sustainability* of knowledge at a human lifespan.

2.2. Phase-1: Fragmentation and Centralization

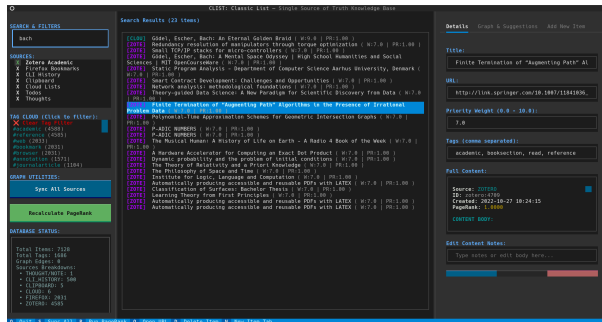


Figure 1: The terminal-based interface of `clist`, with unified search and enriched metadata.

A sample workflow is shown in Figure 1. While `clist` demonstrated the power of unified data sources and enriched metadata, the project was impeded by the limitation of accessing various cloud data sources. However, the project took an interesting turn when even an offline data source turned out to be obstinate: the “list of files” that exist in our filesystem. One direct approach was to reference files within the centralized database of `clist`, however, for longevity of data model (as discussed in the next section), a decentralized approach turned out to be a much better choice. In such a model, every document (file or otherwise) manages its own metadata, and there is no central database. We now formally define the Decentralized Model in Definition 1.

Definition 1 (Decentralized Model). A decentralized model of document management is where documents’ organizational structure and metadata is not managed through a centralized database, rather each document carries its own metadata with itself.

2.3. Phase-2: Longevity/Sustainability and Decentralization

When it comes to the longevity of knowledge organization, filesystem based organization tends to outlive specialized applications[15]. However, the hierarchical model of filesystems pose challenges summarized in Table 1.

To achieve decentralization, we developed the LIST Filesystem Wrapper (LFW). LFW attaches a dedicated metadata file with each file, and gives file an immutable ID. This creates a “laundry-basket” filesystem where documents are stored in a flat structure but logically represent a graph model.

2.4. Phase-3: Formal LIST Specification

As LFW reached a usable stage, we formalized the project into the LIST specification, which is now discussed in the next section.

¹To provide us insights into the problems and solutions pertaining to knowledge management, we mined data from high-quality internet forums like Obsidian, org-mode, Zotero, and Zettelkasten. The data for the first two forums was mined completely (as of 01 May 2026). This data is available in our Git repository: <https://gitlab.com/thelistproject/forumdatamining>.

The project started in 2022 around the problem of *Fragmentation*, and solving it through *Centralized* databases. A normal computer user today has to manage an increasingly large number of documents and document types. Consequently, each user project gets *Fragmented* across multiple applications. Users end up maintaining many disjoint “lists” of files, references, playlists etc.[14]

To unite these lists, we built `clist` (aka, Classic LIST — to disambiguate from the “LIST” specification which emerged later): a terminal-based software that stores disparate knowledge sources in a centralized database. A sam-

Table 1
Challenges of Hierarchical Models and its solutions in LFW

Dimension	Hierarchical/Systemic Limitation	LFW Systemic Solution
Structural	<i>One-file-one-folder</i> constraint.	Multi-location support: A single file can appear in multiple folders via metadata.
Referential	Identity is conflated with location (path-based).	Persistent identity: Unique IDs ensure references remain intact regardless of path.
Cognitive	Requires upfront naming/placement decisions.	Naming flexibility: Renaming does not break links based on IDs.
Persistence	Local identifiers (inodes) are lost during migration.	Embedded persistence: Metadata remains unchanged during transport.

3. The LIST Specification and LFW

3.1. LIST Specification

The *LIST* (Linked Immutably with Static&Strong Typing) specification is built upon two core axioms:

1. **Linked Immutably:** References between units of knowledge (called “Atoms”) are based on unique, immutable identifiers ($\text{id}(a)$) rather than volatile filesystem paths, decoupling an object’s identity from its name or location.
2. **Static and Strong Typing:** Each “Atom” is “strongly” and explicitly (hence “statically”) associated with a metadata object M_a .

Structurally, a *LIST Universe* $\mathcal{U}$ is a set of these Atoms and the immutable directed edges (links) between them. This model supports recursive composition, enabling granular referencing that remains stable regardless of how the Atoms are renamed or reorganised.

Note that the notion of Static and Strong typing has been abstracted from Types in Programming languages. Much like a static language has explicit information (metadata) about each of its variables, and a strong language doesn’t easily lose that information (metadata), LIST specification promotes explicit and rich metadata, and that metadata isn’t easily lost (hence preserving file provenance). This is briefly described in Table 2. More details about this specification are available on project website, and an initial formalization of this specification in Haskell is available in our [Git Repository](#).

Table 2
Programming Types and the LIST Type System

Paradigm	Programming	FileSystem
Dynamic/Loose	$a = 45$ (Type is inferred/mutable)	$f.\text{txt} \rightarrow \langle \text{FileContent} \rangle$ (Metadata is minimal/volatile)
Static/Strong/LIST	$\text{float } a = 45$ (Type is explicit/fixed)	$M_a f.\text{txt} \rightarrow \langle \text{FileContent} \rangle$ (Metadata is explicit/preserved)

3.2. LFW Filesystem

To demonstrate this specification, we developed LFW filesystem. LFW works through a two-step process:

- **Conversion:** In this step, the traditional hierarchical data is flattened into an *LFW root*, where each file is assigned a UUID and paired with a corresponding `.md2` metadata file to preserve provenance. Each entity in LFW is classified in the following two types during conversion:

²Kindly note that these design choices, such as choosing `.md` as metadata extension, are temporary.

- **Bundles:** Directories that must be preserved as a single unit to maintain internal dependencies - for example, Git repositories or folders explicitly marked with a `.bundle` extension.
- **Singletons:** Files that don't have strong dependencies and can be treated as independent.
- **Mounting:** Once the conversion is done, LFW constructs a graph-based index of these Atoms, allowing the user to project the data back into a familiar hierarchical view or alternative organizational structures without altering the underlying immutable IDs.

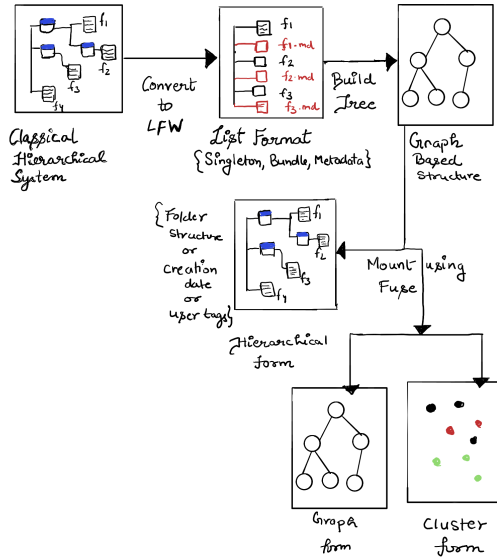


Figure 2: LFW Architecture

Crucially, LFW is designed for *reversibility*, ensuring that users can transition back to standard hierarchical storage without loss of data or metadata, thereby avoiding vendor lock-in. LFW architecture is shown in Figure 2, and further implementation details and benchmarking is available on the project website.

3.3. LFW Application

In this section, we discuss one interesting application of LIST: `git_lfw`. One issue programmers often face is moving and renaming files during refactoring. However, the version control systems like git often lose track of these changes and instead considers them to be a deletion of the old file and the creation of a new file. With LFW, we were able to create a git middleware that seamlessly allows users to work with git while at the same time preserving the file's modification history. This is done using a hidden `.lfw` directory that stores the immutably identified files and metadata that git actually tracks, while the middleware translates to the hierarchical view the users expect. For more details, you may refer to its current implementation in Python alongwith a video demo on our Git repository.

4. Conclusion

In this work, we introduced the LIST conceptual model (specification) for knowledge management, and its reference implementation as a filesystem – LFW. Furthermore, the application of this model in `git_lfw` illustrates how this model can solve long-standing issues of provenance in areas beyond knowledge management. Our immediate focus ahead is to bring LFW to a stage where non-technical users can start using it. Future work entails several exciting possibilities, like developing a new alternative to Git, developing a new document format that abides by the LIST model, and using LFW for user-behaviour analysis (for knowledge management research).

Finally, we add a note about the key motivation of this project: *Sustainability*, which we intend in two ways here. **One** is the sustainability of the knowledge management for regular users at a human lifespan; this aspect we have already discussed in this paper. **Two** is the (indirect) aspect of environmental sustainability. The 2024 documentary “How AI was stolen”[16] brought an awareness to the world about how the current Artificial Intelligence (AI) systems are built on stolen data from human knowledge, and several researchers [17] had warned us about the vast environmental consequences that we will incur of AI model training and inference. And this stealing shall continue over time, incurring more stress on the environment. We hypothesize that a good data organization model for the users will preclude them

from handing over their data in the hands of the corporate cloud services, and hence it can serve as a moat to the centralization of AI. This may seem like an unrelated consequence of this project, but it served to be the most motivating factor for the authors.

Declaration on Generative AI

This paper is completely human-written in Emacs (in Evil mode) with no use of Generative AI. The code available in our main repository (LFW) is also pure human.

References

- [1] T. Mason, Laundry Baskets: The New File System Namespace Model, 2021. URL: <https://fsgeek.ca/2021/09/28/laundry-baskets-the-new-file-system-namespace-model/>.
- [2] W. A. Mason, Indaleko : The Unified Personal Index, Ph.D. thesis, University of British Columbia, 2025. URL: <https://doi.library.ubc.ca/10.14288/1.0449905>.
- [3] S. Gruner, Heinz zemanek's almost forgotten contributions to the early philosophy of informatics, in: Australasian Conference on Information Systems, 2016.
- [4] W. Kent, Data and Reality 2nd Edition, rev ed., 1st Books Library, Bloomington, Indiana, 2000.
- [5] L. Alon, R. Nachmias, Gaps between actual and ideal personal information management behavior, *Computers in Human Behavior* 107 (2020) 106292. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0747563220300480>. doi:10.1016/j.chb.2020.106292.
- [6] E. Thereska, O. Riva, R. Banks, S. Lindley, R. Harper, Beyond File Systems: Understanding the Nature of Places Where People Store Their Data, Technical Report, MicrosoftResearch, 2013.
- [7] M. Geel, Memsy: A Personal Resource Management Infrastructure, Ph.D. thesis, ETH Zurich, 2015. URL: <http://hdl.handle.net/20.500.11850/113707>. doi:10.3929/ETHZ-A-010603237. arXiv:20.500.11850/113707.
- [8] L. F. Lemos, D. Castro, C. E. Pantoja, B. Freitas, Tag-Driven File Management System, *Revista Eletrônica de Iniciação Científica em Computação* 24 (2026) 155–161. URL: <https://journals-sol.sbc.org.br/index.php/reic/article/view/6680>. doi:10.5753/reic.2026.6680.
- [9] Nayuki, Designing better file organization around tags, not hierarchies, 2017. URL: <https://www.nayuki.io/page/designing-better-file-organization-around-tags-not-hierarchies#contents>.
- [10] Karl Voit, Novoid/filetags: Management of simple tags within file names, 2013. URL: <https://github.com/novoid/filetags>.
- [11] K. Voit, TagTrees: Improving Personal Information Management Using Associative Navigation, Ph.D. thesis, Graz University of Technology, 2012.
- [12] M. Seltzer, N. Murphy, Hierarchical File Systems are Dead, in: *Usenix*, 2009.
- [13] Shivam Sharma, Github:shivams/zim-zotero-plugin, 2017. URL: <https://github.com/shivams/zim-zotero-plugin>.
- [14] Jeff Atwood, Reinventing the Clipboard, 2008. URL: <https://blog.codinghorror.com/reinventing-the-clipboard/>.
- [15] J. D. Dinneen, C.-A. Julien, The ubiquitous digital file: A review of file management research, *Journal of the Association for Information Science and Technology* 71 (2020) E1–E32. URL: <https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.24222>. doi:10.1002/asi.24222.
- [16] Lewis Waller, How AI was Stolen, 2024. URL: <https://www.youtube.com/watch?v=BQTXv5jm6s4>.
- [17] E. M. Bender, T. Gebru, A. McMillan-Major, S. Shmitchell, On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? , in: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, ACM, Virtual Event Canada, 2021, pp. 610–623. URL: <https://dl.acm.org/doi/10.1145/3442188.3445922>. doi:10.1145/3442188.3445922.