TagFS — Tag Semantics for Hierarchical File Systems

Stephan Bloehdorn
Institute AIFB, University of Karlsruhe
D-76128 Karlsruhe, Germany
sbl@aifb.uni-karlsruhe.de

Max Völkel
Institute AIFB, University of Karlsruhe
D-76128 Karlsruhe, Germany
mvo@aifb.uni-karlsruhe.de

ABSTRACT

Today, most computer users work with traditional hierarchical file systems for organizing large amounts of personal files. Recently, *tagging* has grown popular as an alternative means of organizing information resources, especially in collaborative contexts.

This paper analyzes the different semantics between strictly hierarchical and tagging-based organisation. Based on these observations, we map non-hierarchical tagging and query semantics to the commonly used hierarchical file system semantics, thus combining the benefits of both worlds. Our WebDAV-based implementation allows users to collaboratively manage their files in a tag-based manner via existing, familiar file system explorers.

1. INTRODUCTION

The amount of digital information stored on single computers is increasing steadily [1]. Most users are familiar with the hierarchical file system of their operating system which they efficiently browse for organizing the large amount of available files. The major building block of a hierarchical file systems is the *directory* as an organizational unit which acts as a container for files or further (sub-)directories. A structure of nested directories is commonly used for a sophisticated organization of files according to the semantics of their content from most general at the top level to most specific at the leaf levels of the directory hierarchy.

Example 1 (Music Files). *A typical organization pattern for music files is to maintain different directories for each artist at the root level, further subdirectories for the different albums in which the actual files are situated using the track title as the file name.*

Example 2 (Scientific Papers). *A typical organization pattern for storing digital versions of scientific papers of interest would be to maintain different directories for different research areas possibly including refinements for different scientific subfields.*

WWW2006, May 22–26, 2006, Edinburgh, UK.

Unfortunately, traditional hierarchical file systems don't serve their purpose well. Barreau and Nardi [2] observed in a study, that "Every user [...] indicated that their attempts to establish elaborate filing schemas for archived information failed because they proved to require more time and effort than the information was worth." We found a number of reasons for the inability to help people organise information resources with file systems:

Single Location A file can reside at only one location in a strict directory hierarchy at the same time without being duplicated[1]. In Example 2, a scientific article—unless being duplicated—can only reside within one of the two directories `ir` or `semweb` although both containers might be semantically adequate if the paper is dealing with both topics, *information retrieval* and *semantic web*, at the same time.

In fact, the *single location* property entails a number of deficiencies which can be specified in more detail:

Browsing to Maximum Specificity Retrieval of a file by browsing requires navigation to the exact directory where it is stored. An implicit classification of the file object into higher-level directories along the path is not done. In Example 1 this would for example require a user in the search of Lisa Ekdahl's "Stranger On Earth" to navigate the full path down to the album "Back To Eearth", while a search in the "Lisa Ekdahl" top-level directory will yield no results.

Missing Orthogonality Files are classified by different dimensions. E. g. digital pictures can be organized by time, by topic, by location, or by persons which appear in the picture. All these orthogonal dimensions have to be squeezed into a single hierarchy, that allows only one access path.

Path Order Dependance While the directories along a path are often perceived as independent attributes, the semantics of the hierarchical file system contrasts this by a highly order-dependant interpretation. In Example 2, a directory structure containing the two paths `ir/semweb` and `semweb/ir` may grow naturally over time, which point, however, to entirely different sets of files.

[1]Although some file systems allow a file to reside at multiple locations simultaneously (via *hard links*), most of the problems described below remain or require a lot of manual effort compared to tagging.

Independent of the *single location* issue and its implications, traditional file systems exhibit two further deficiencies:

No Query Refinement The file system offers no dynamic refinement strategy in browsing for files: each directory contains only directories that have been explicitly put there. No other potentially relevant directories to search in are listed.

No Navigational Aid All directories within a directory look the same, with no indication of the content of sub-folders, e. g. regardless whether they are almost empty or contain large sub-hierarchies.

Recently, *tagging* has become an alternative approach for semantically organizing information resources, which has grown in popularity especially in the context of *collaborative* tagging systems as for example del.icio.us[2] for bookmarks or flickr[3] for digital images.

In such tagging systems users are annotating information resources with freely chosen keywords. All items tagged with the same keyword implicitly form a set. A set of arbitrary tags is often treated as a conjunctive query, denoting the intersection of the tag-induced sets. Consequently, tags or tag combinations, and the sets they imply, become the major organizational unit of tag-based information management. Information organization based on tags does not suffer from the problems identified for information organization in hierarchical file systems.

The main contribution of this paper is a clean mapping of non-hierarchical tagging and tag-query operations to existing hierarchical file system operations. We present TagFS, a *file system with tag semantics*, as a means for managing, browsing and retrieving large amounts of files efficiently. On the first sight, the resulting system behaves like a traditional file system but the retrieval and change operations like `copy` or `move` carry tagging based semantics.

Outline

This paper is structured as follows: In section 2 we contrast the two approaches for organization of information resources in more detail and present a mapping of tag-based systems to the traditional file system paradigm by assigning new semantics to the traditional file system operations. In section 3 we outline the design and implementation of a prototype which implements the tagging semantics behind a WebDAV-compatible abstraction layer. In section 4 we point to related work on semantic file systems and we conclude in section 5.

2. DESIGN

In this section, we briefly review the structural properties of hierarchical file systems and tagging based systems, then explain how we redefine folder operations to tag operations and browsing to conjunctive queries.

2.1 Classical File Systems and Tagging

Hierarchical File System. An organizational unit of *information content* stored within the file system as a sequence of bits makes up what we regard as a *file*.

Typically, an index structure (the file allocation table) provides a mapping from a set of *access paths* to the corresponding blocks of information. As such, the access paths act as *keys* for accessing the file content. The perceived organization of files into directories is a result of the structure of the access paths.

Files are accessed via an *access path*. Each access path can be decomposed into a sequence of directories (which we will call the *location*) and a *file name* at the end. E. g. we decompose the access path at `/paper/2003/semweb/gruber03.pdf` into *location* = `paper`, `2003`, `semweb` and *file* = `gruber03.pdf`.

Tagging Systems. The actual storage of an information item in a tagging system is typically opaque to the user. The file content is accessed by means of a *resource key*[4]. On the tagging layer, *tag annotations* are freely attached to resources. Formally, tag annotations can be seen as unary predicates applied to the resources. As such, tag annotations imply different *sets* of resources which can be combined using the standard set operations of union, intersection and complement. In the following, we will restrict our focus on *conjunctions of tags*, i.e. intersections of the corresponding sets, which we will call *query* or—in analogy to the hierarchical file system—*location*. Note that in this paradigm, one and the same resource may reside within different locations at the same time.

2.2 Mapping Tagging Semantics to Hierarchical File Systems

The following section describes the mapping between the tag-based and the hierarchical file system approach on the basis of three use cases: retrieval, re-organisation, and adding files to TagFS.

In brief, *tags are assigned or changed by moving, copying and deleting files.* New files are added by copying or moving from an external store to TagFS. File deletion and tag creation require special workarounds, as described below.

Retrieval: Mapping Locations To Queries. In our mapping, directories are interpreted as tags and locations (sequences of directories) are interpreted as conjunctions of tags. We define the two functions *view(location)* and *subfolders(location)* for browsing and retrieval. In a file system, the view of a directory is the union of both functions.

view(location) returns all files, or – strictly speaking – resources, that have been tagged with all tags contained in the location. Thus we treat folders as tags and files in a location represent the query results. E. g., in Example 1 a view of the folder `/lisa-ekdahl/swedish-song/2005` shows all music files that carry the tags `lisa-ekdahl`, `swedish-song` and `2005` (and possibly other additional tags as well!) with the obvious interpretation of these tags. Note that the folder view for a different permutation of these tags as in `/2005/swedish-song/lisa-ekdahl` would return the very same set of resources as it corresponds to the equivalent conjunctive query.

[2] `http://del.icio.us`
[3] `http://www.flickr.com`

[4] The resource key may in fact be (and often is) a simple access path on a lower "storage" layer.

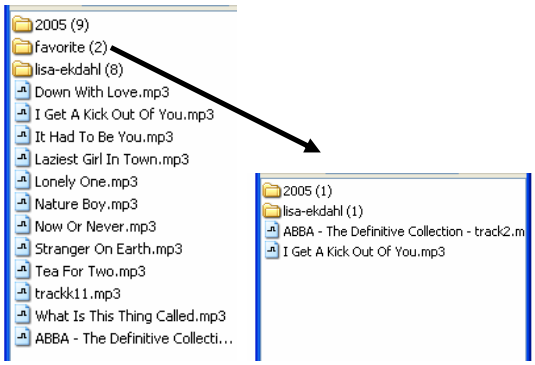

Figure 1: Refining a query by browsing

subfolders(location) returns all those tags as subdirectories, which can be used as a refinement of the current conjunctive query without returning an empty query result. In Example 1 (c. f. Fig. 1) imagine that the user had tagged Lisa Ekdahl's song "Now Or Never" using the tags `lisa-ekdahl`, `swedish-song` and `2005` and `favorite`. The song resource would have been displayed in the previous example on the location `/lisa-ekdahl/swedish-song/2005`. The `subfolders(/lisa-ekdahl/swedish-song/2005)` would (probably amongst others) return the tag / subfolder `favorite` as an indication of a possible refinement. Similarly `subfolders(/2005/favorite)` would return `lisa-ekdahl` and `swedish-song` as subfolders. Each directory is named like the tag name plus the number of query results, like '2005 (847)' and 'favorite (8)'.

Organization: Mapping File System Operations to Tag Changes. We use the following mapping from file system to tagging semantics, roughly mapping files to resources and tags to folders.

delete(location,file) removes the last tag from the specified file. This operation is consistent with the traditional file system in that the file will no longer occur in the specified folder.

In order to really delete a file, it has to be tagged with `delete`. This results in the immediate removal of the specified file from the storage backend and removal of all tagging references to this file.

copy(location source,location target,file) assigns additional tags from the target folder to the file in question. Note that this will in general result in additional results for the subfolders(...) operation on the source folder.

move(location source,location target,file) consistent with the *delete(...)* and *copy(...)* operations, this operation removes all the tags denoted by the source location and assigns all the tags denoted by the target location to the file in question. This behavior is again consistent with the behavior of traditional file systems in that the file disappears from the old location and appears in the new location.

createDirectory (tag) Creating a new directory is only necessary if no file carries the corresponding combination of tags so far. However the newly created directory would not be shown at the location it was created, because only tags with non-empty query results are listed. We tackled this problem by means of the following heuristic: TagFS uses a special placeholder file `just_created`, tagged with all the tags implied by the current location plus the newly introduced tag. The placeholder file will remain tagged until an actual file receives the relevant tags, the user session has ended or a timeout has occurred. If the corresponding tag does not yet exist, it is created.

move, copy or delete a directory These commands are executed for each file at source location.

Adding Files: Mapping Files to Resources. Resources, by definition, unambiguously identify file contents. The file name as the last part of the traditional access path can, taken by itself, be ambiguous in the context of a tag based file system, because the root folder lists all files in one directory.

We thus require unique filenames. For presentation, we change the filename at insertion time by adding a disambiguation number into the file name just before the file type extension. In our previous example this would result in `gruber.pdf` and `gruber-1.pdf`. For each file, we store the unique MD5 hash over the file content as resource keys. This approach results in the following operation for the new file system:

put(external-acces-path,location) this operation results in the following actions: (i) insertion of the external file content into the storage backend, uniquely mapped to a resource key equivalent to the MD5 hash; (ii) assignment of all tags denoted by the location to the resource key; (iii) assignment of the file name composed of the original filename, a disambiguation number and its file type extension to the resource. E. g. in Example 2 moving a paper with filename `gruber93toward.pdf` from outside TagFS into the folder `/1993/ontology` would assign the tags `1993` and `ontology` to the newly created resource `gruber93toward.pdf` .

This approach brings in certain peculiarities as for example that a second file with the same content hash as a previously stored file is considered to be the same file, regardless of original file name. This enables files which are copied from the semantic file system to another store (e. g. email attachment) to be identified as the same file, when re-inserted it into the semantic file system.

3. IMPLEMENTING TAG SEMANTICS FOR HIERARCHICAL FILE SYSTEMS

TagFS is implemented in Java as a virtual file system, exposed via WebDAV. The overall architecture is depicted in Fig. 2. On the highest level, we have different users, interacting with their existing file explorers over the WebDAV protocol, from which we use the specifications [3] and [4]. For implementing a WebDAV server we use two open source

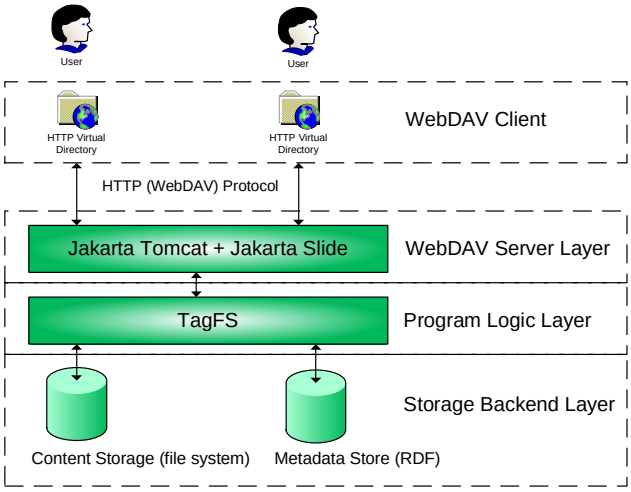

Figure 2: TagFS System Architecture

tools from Apache Jakarta, namely *Tomcat* and *Slide*. In the very heart of the system, we implement our business logic (the mapping). Further down, we use two different storage back-ends: A classical file system, storing the binary file contents and an RDF triple-store (a specialized RDF database), storing the tag assignments in RDF format.

In the next section we explain why we chose WebDAV, how we manage the tag assignments and how we store files. The mapping itself has been described in Sec. 2.

Integration of a Virtual FS. In order to allow users to seamlessly interact with TagFS using their standard file explorers like the *Microsoft Windows Explorer*, a commomny supported protcol needs to be used.

The windows network file system protocol SMB/CIFS often does not work across local network boundaries because of firewalls. Furthermore there is no Java implementation for it.

FTP does not support secure transmission and authentification and secure FTP (SFTP) is badly integrated in MS-Windows and Mac standard filesystem tools. Even worse: A simple operation such as a remote move command is modelled as download–move–upload, resulting in a waste of time and bandwidth.

WebDAV solves these problems. It is an HTTP-based protocol for remote file system operations. Remote WebDAV-folders can be mounted as Windows network folders (built-in support) and can also be mounted in Linux and Mac OS file systems.

We decided to use a WebDAV sever as the interface to the client (Windows Explorer, Unix file system mount). The backend for the actual file storage is explained in the next section.

Managing Tag Assignments. We manage tags using RDF, the *Resource Description Framework*. Each tag assignment is stored as a set of three RDF triples: It has a file, a tag and a user. This enables collaborative tagging. We also store the original file information such as file size and creation date in the RDF store. As direct manipulation of RDF data is rather cumbersome, we use RDFReactor [5] to map our RDF

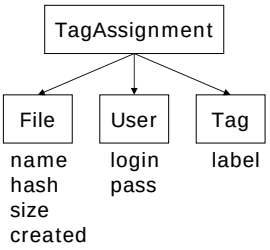

Figure 3: Tag Assignment Schema

Schema[5] to Java classes.

Storage Backend. As the internal file storage is never exposed to the user, we have many design options. For simplicity and backup reasons we choose to store files in a classic file system directory. As current file systems do not perform well with very large folders, we use a two-level index strategy, e. g. storing a file `abcde.txt` as folder `a/ab/abcde.txt`.

4. RELATED WORK

A study on personal file management [2] states a strong preference of users to guess file locations and browse for a file in a list rather than using keyword search tools. Often they find their files in the first or a few tries. To achieve this, users consciously organize their files for easy retrieval, using directories to relate files with a work context.

There are some works which try augment file systems with additional semantics. Back in 1991, Gifford et. al. [6] list about ten implementations of *virtual file systems* in UNIX. A virtual file system is described as a set of *virtual folders* which appear within existing tree structured file systems. A *virtual directory* abstracts storage locations away and lists files based on a query which determines the content dynamically.

Gifford's system extracts attributes, modeled as key-value pairs, from file via a set of *transducers*, one for each file type. He maps virtual directory names to queries, using one path entry for the key and the following for the value. E. g. `/location/London/author/Max` would list all images showing London, shot by Max. Gifford distinguishes between normal and hidden virtual directories, to remain compatible with existing network file system protocols and file archiving tools such as *zip* and *tar*.

The work of Soules and Ganger [7] is quite similar, but emphasizes the aspect of getting usable metadata. They observe, that users do not like to state additional information when saving a file to a directory, but are willing to choose a directory and name for the file. Both directory and filename can be seen as a context of the file, which can be stored as metadata.

The more recent work of Mills [8] describes an implementation which combines the ideas of [6, 7] and extracts EXIF and ID3 metadata from digital pictures and music files. When a user puts a file into `location/London` the appropriate EXIF information is also *written back* into the file. New attributes and possible values are created by creating new directories. Mills builds upon [9], which runs only on

[5] `http://semfs.ontoware.org/2005/tagfs`

UNIX systems.

An extension [10] to the popular browser Firefox allows users to browse their del.icio.us tags as a virtual hierarchy. However, the support is read-only and only two levels of tags are rendered as nested folders.

Our Contribution. TagFS steps away from attribute-value-pairs and offers only unary tags. This makes the transition from classical file systems to TagFS easier. In key-value-based systems, the folders denoting keys are not allowed to contain files, resulting in hard to understand error messages for users. TagFS is platform independent and—thanks to WebDAV—works across firewalls. Thus TagFS is more suitable for collaboration. Finally, TagFS makes tag authoring much easier.

Soules and Ganger [7] stress the need to keep relations between files. TagFS makes this easy: users simply add another tag to form a new group. This does not affect previous ways to find the files.

TagFS makes the explicit, manual management of tags so simple, that users don't have to rely on automatically—and possibly error-prone—extracted meta-data. They are also not forced to use a fixed structure of keys. Instead, TagFS users can choose their keywords freely and evolve their organisation scheme as needed.

5. CONCLUSION AND OUTLOOK

In this paper we presented TagFS, a virtual file system with tagging semantics. TagFS allows users to manage their files collaboratively through their existing desktop applications for file management, provided WebDAV support. We explained how we overload file system operations with tagging semantics. Then we sketched our current system architecture and discussed related work.

Until now, no user study on virtual file systems is known to the authors. We expect TagFS to be as easy to use as current tagging systems on the web. Simple tasks like "give all papers on machine learning and semantic web to Stephan" boil down to either copy the directory `ml/semweb` or grant Stephan access to a shared TagFS server. Compared to existing tagging systems TagFS has its strength on tag assignment and management, in particular mass-tagging via drag and drop is offered.

Outlook. We identified a number of directions for future work, which we list briefly:

- Export all meta data as RDF, enabling others to come up with even better ways to re-use, visualise, search and browse the emerging structure
- Management of del.icio.us and flickr tags within TagFS
- Systems tags like `_older-than-2-weeks` or `_bigger-than-1MB` could be generated automatically.
- Extracting metadata (EXIF, ID3) from files automatically could also enhance usability.
- User Management: WebDAV can require the user to login before performing any operation. This enables per-user tracking of all operations and enables more sophisticated collaborative tagging.

Acknowledgments: This research was partially supported by the European Commission under contracts IST-2003-507482 Knowledge Web (`http://knowledgeweb.semanticweb.org`), IST-2006-027705 NEPOMUK and IST-2003-506826 SEKT (`http://www.sekt-project.org/`). The expressed content is the view of the authors but not necessarily the view of the project consortia. We would like to thank our colleagues for fruitful discussions, Andreas Kreidler for assistance in implementation work, and Heiko Haller for help in editing this paper.

6. REFERENCES

[1] Lyman, P., Varian, H.R.: How much information (2003) `http://www.sims.berkeley.edu/how-much-info-2003`, retrieved on 02.11.2005.

[2] Barreau, D., Nardi, B.A.: Finding and reminding: file organization from the desktop. SIGCHI Bull. **27** (1995) 39–43

[3] Goland, Y., Whitehead, E., Faizi, A., Carter, S., Jensen, D.: Http extensions for distributed authoring – webdav. Technical report, The Internet Society (1999)

[4] Clemm, G., Reschke, J., Sedlar, E., Whitehead, J.: Web distributed authoring and versioning (webdav) access control protocol. Technical report, The Internet Society (2004)

[5] Völkel, M., Sure, Y.: Rdfreactor - from ontologies to programmatic data access. Poster and Demo at ISWC2005 (2005)

[6] Gifford, D.K., Jouvelot, P., Sheldon, M.A., James W. O'Toole, J.: Semantic file systems. In: SOSP '91: Proceedings of the thirteenth ACM symposium on Operating systems principles, New York, NY, USA, ACM Press (1991) 16–25

[7] Soules, C.A.N., Ganger, G.R.: Why can't i find my files? new methods for automating attribute assignment. In Jones, M.B., ed.: HotOS, USENIX (2003) 115–120

[8] Mills, B.: Metadata driven filesystem (2005)

[9] Mazières, D.: A toolkit for user-level file systems. In: Proceedings of the General Track: 2002 USENIX Annual Technical Conference, Berkeley, CA, USA, USENIX Association (2001) 261–274

[10] Ayala, D.: Foxylicious - firefox and del.icio.us bookmark integration (2005)

Short Bios

Stephan Bloehdorn

Stephan Bloehdorn is researcher at Institute AIFB at the University of Karlsruhe. He received his Master from the University of Karlsruhe after studying Information Engineering and Management and is currently pursuing a Ph.D. Stephan's research interests include machine learning on semantic web data, text mining, knowledge representation, user interaction and knowledge management. Stephan is currently working in the EU IST Project SEKT. Recently, Stephan has co-organized the ICML 2005 Workshop on Learning in Web Search.

Max Völkel

Max Völkel is a researcher at the Institute AIFB at the Universität Karlsruhe, where he is working on his Ph.D. He did his Master in Computer Science at the Universität Karlsruhe. His research interests include personal knowledge management, semantic wikis, knowledge representation and Semantic Web infrastructure. Max is currently working in the EU IST Project NEPOMUK. He is currently organising a workshop on semantic wikis at the ESWC 2006.