Implementing Filesystems by Tree-aware DBMSs

Alexander Holupirek
supervised by: Marc H. Scholl

University of Konstanz
Dept. of Computer & Information Science
Box D 188, 78457 Konstanz, Germany

holupire@inf.uni-konstanz.de

ABSTRACT

With the rise of XML, the database community has been challenged by semi-structured data processing. Since the data type behind XML is the tree, state-of-the-art RDBMSs have learned to deal with such data (*e.g.*, [18, 5, 6, 16]). This paper introduces a Ph.D. project focused on the question in how far the tree-awareness of recent RDBMSs can be used to, once again, try to implement filesystems using database technology. Our main goal is to provide means to query the data stored in filesystems and to find ways to enhance/ combine the data storage and query capabilities of operating systems using semi-structured database technology.

Two DBMSs with *relational* XML storage, built on top of the XPath accelerator numbering scheme [14], are the foundations for our work. With BaseX, an XML database, we establish a link between user, database and filesystem content. BaseX allows visual access to filesystem data stored in the database. An integrated query interface allows users to filter query results in interactive response time. Second, we establish a link between DBMS and OS. We implement a filesystem in userspace backed by the MonetDB/XQuery system, a well-known relational database system, which integrates the Pathfinder XQuery compiler [5] and the MonetDB kernel [4].

As a result, the DBMS is mounted as a conventional filesystem by the operating system kernel. Consequently, access via the established (virtual) filesystem interface as well as database enhanced access to the same data is provided.

1. INTRODUCTION

Since the beginning of database management systems, there is a desire to store all data in a database and have it ready to be queried. Several industrial and research efforts such as WinFS or the Be Filesystem have been undertaken to push the filesystem into a database. None made it to technical production quality. Offshoots, like Microsoft's Instant Search or Apple's Spotlight Architecture, however, can be found in all of the recent operating system variants, and a user demand for products helping to find relevant content can be derived from the increasing popularity of Desktop Search Engines, such as Google's or Yahoo's Desktop Search. While these tools offer a smarter way to access personal information stored in the filesystem, the keyword-driven search approach, as it is used by today's search engines, is inherently limited. An additional support for database style query languages would be preferable.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-306-8/08/08

1.1 Problem Description

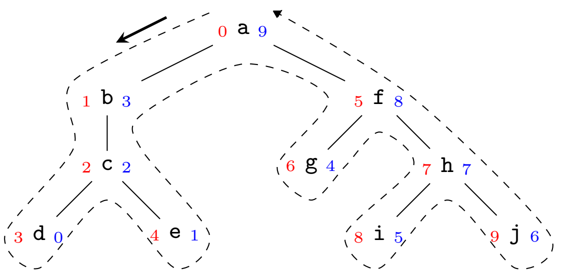
We generally face the fact that the amount of data stored in filesystems on personal computers is growing steadily. This comes as no surprise since—against current opinion—data gets copied from old machines to new ones instead of being curated, archived and purged from the working system. This may be considered a bad habit, but it surely is a side effect of storage capabilities increasing at low cost, and thus cannot be condemned. Jim Gray et al. pointed out [12] that a "decade ago, 100 GB were considered a huge database. Today it is about 1/2 of a disk drive and is quite manageable. For example, a thousand 1990's magnetic tapes fit on a single online disk today—so it is both economical and desirable to bring the old data forward and store it on newer technology." Therefore filesystems contain a significant amount of text documents, images, and multimedia files. While the mere storage is an easy-to-manage task, convenient access to and information retrieval from huge amounts of data is crucial to leverage the stored information. Current filesystems and their proven, but basic interface (VFS) support neither of it.

1.2 Challenge

Donald Norman coined the phrase "Attractive things work better" [26]. While Norman's statement, in first place, aimed at pushing aesthetics and attractiveness into user interfaces, it suits well for any human-centered design approach. Without usability, joy of use cannot evolve. Ease of use, on the other hand, is crucial and for a data storage system it is determined by the ability to search/find and access/use stored data. In fact, the challenge we now face (and will even more in the future) is to enhance storage systems in a way that users can make full use of their data. Finding relevant content in this ever growing amount of data is a major aspect. Filesystems still focus on mere storage and tend to be conservative regarding feature enhancements [33]. Consequently, they do not offer solutions to this demanding task.

```
$ tree ./a
0a9
|-- 1b3
|   `-- 2c2
|       |-- 3d0
|       `-- 4e1
`-- 5f8
    |-- 6g4
    `-- 7h7
        |-- 8i5
        `-- 9j6
```

```
0<a>
 1<b>
   2<c>
     3<d/>0
     4<e/>1
   </c>2
 </b>3
 5<f>
   6<g/>4
   7<h>
     8<i/>5
     9<j/>6
   </h>7
 </f>8
</a>9
```



pre	post	n
0	9	a
1	3	b
2	2	c
3	0	d
4	1	e
5	8	f
6	4	g
7	7	h
8	5	i
9	6	j

Figure 1: Basic (simplified) idea of storing trees (such as file hierarchies, XML documents) in a RDBMS [14].

1.3 Research Approach

Despite the fact that several database-driven filesystem attempts have already failed, the advent of XML brought some significant enhancements to (R)DBMS that inspired us to dare another attempt.

In a preliminary study to this Ph.D. project (published in [22]), we have evaluated the mapping of a file hierarchy and its content to XML and emulated filesystem operations using XPath/XQuery/XQUF operations. We found it possible to perform basic filesystem commands, as well as content-based retrieval, in interactive time on the constructed filesystem mappings with an off-the-shelf XML database. Motivated by these results we pushed the idea forward.

The tree-based XML model has spawned efforts on relational storage and processing techniques for hierarchically structured data and meanwhile, RDBMSs have learned to work with tree-shaped data (*e.g.*, [18, 5, 6, 15, 16]). This is of direct benefit, as the hierarchic nature of filesystems can now consistently be mapped to the relational storage (see Figure 1) and leverage the associated algorithms (an elaborate discussion of relational XML storage and algorithms can be found in [31, Chap. 2]).

A major problem of storing files in a DBMS (apart from using BLOBs) has been the basic necessity of providing a schema first. With an unmanageable amount of file formats this appears to be an impossible mission. Schema-oblivious storage techniques rendered it possible for XML data to be stored in the database without previous knowledge of its interior structure[1].

Textual files can easily be converted or wrapped into XML. More and more applications use XML as their native storage format anyway (OOXML, OpenDocumentFormat). Data of this kind is already prepared to be handled with database technology. From our point of view, these documents are nothing else but serialized database instances. In consequence, they are not only stored as plain text, but directly shredded[2] into the DBMS. Legacy applications are still able to process them conventionally by requesting them in their serialized, *i.e.*, textual representation. In direct communication with the database, however, XML processing languages such as XPath/XQuery can be used on the data. Going through with the concept, and as filesystems are structured hierarchically, it seems to be a natural thing to also map the file hierarchy into tree-aware DBMSs.

[1]An additional XML Schema specification for the file type may be of advantage to formulate queries against the document, but is not mandatory.

[2]A terminus technicus used to indicate the conversation of XML in its textual representation to an internal format used by the database. Read it as “import”.

1.4 Outline

This paper presents work at two opposite ends of a suitable DBMS architecture. We will dig down and contribute an implementation that establishes a link between DBMS and OS and show how a database with XML/XQuery support can be used as a user level filesystem. Afterwards, we move up to the database frontend and demonstrate how visual access and interactive querying on filesystem data, stored in the DBMS, can turn databases into primary processors for personal information management systems. Related work and a summary will complete our paper.

Please note that this is work in flux and marks the current state of our research. Since there has been an explicit call for *students who are in the beginning stages of their work* we aim at providing inspiring thoughts for a lively discussion.

2. THE DATABASE AS FILESYSTEM

Traditionally, files are roughly classified as either text or binary. We add XML as a third type and expose formerly hidden content of files to the system and the user with both, its structure and content. The implementation establishes a link between database and operating system and allows the use of XML processing languages, such as XPath and XQuery, on the data. Since the database is mounted as a conventional filesystem by the operating system kernel, access via the established (virtual) filesystem interface as well as database enhanced access to the same data is provided.

The implementation uses MonetDB/XQuery, a well-known relational database system with XQuery and XQuery Update Facility (XQUF) support, that integrates the Pathfinder XQuery compiler [5] and the MonetDB kernel [4].

2.1 The MonetDB/XQuery System

Pathfinder is an XQuery compiler implementation backed by relational database kernels. It is based on the transformation of XML document collections into relational tables [14] and emits relational algebra plans, which consequently could be executed on any RDBMS [17]. In combination with the MonetDB kernel [4] it results in the open

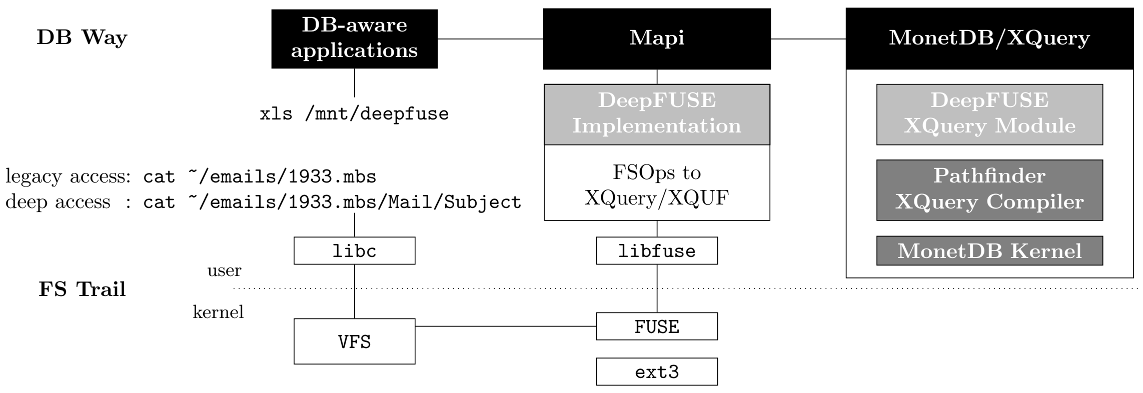


Figure 2: Establishing a link between OS and DBMS by implementing the DBMS as filesystem in userspace. Conventional as well as database supported access to the filesystem data is achieved. Navigation into the file (deep access) is possible by letting the file hierarchy immerse into the file (see also Figure 3).

source MonetDB/XQuery system, which has proven to be one of the fastest and most scalable XQuery implementations available today [6].

RDBMSs incorporate the knowledge and research efforts of years. They have proven to store and query large data sets efficiently and can be considered mature technology. Using the power of relational database technology in combination with recent findings in the domain of semi-structured data processing makes such systems a promising choice for the implementation of a user level filesystem with query capabilities. Besides, the Pathfinder XQuery compiler appears particularly well suited due to the following aspects:

Scalability. It is able to handle huge amounts (up to 10GB) of XML data in an efficient manner.

Targets multiple back-ends. The Pathfinder XQuery compiler has already been enhanced by a code generator that emits SQL. This code generator targets off-the-shelf relational database systems (*e.g.*, DB2) and turns them into efficient and scalable XQuery processors [17, 15].

Integrates a XML information retrieval system. PF/Tijah [21] is a text search system that is integrated with the Pathfinder compiler. It includes out-of-the-box solutions for common tasks such as index creation and result ranking. We expect that to be of great benefit for the stored textual and XML files.

2.2 Filesystems in USErspace (FUSE)

FUSE is a framework for implementing filesystems outside the operating system kernel in a separate protection domain in a user process. It was first implemented for and integrated into the Linux kernel [30]. There are reimplementations for the Mac OS X [28], FreeBSD [20], and NetBSD [24, 25] kernels. The FUSE library interface closely resembles the in-kernel virtual filesystem interface. The user level implementations are able to register function callbacks that get executed once a corresponding request is issued by the OS kernel. The FUSE kernel module and the FUSE library communicate via a special file descriptor: `/dev/fuse`. This file can be opened multiple times, and the obtained file descriptor is passed to the mount syscall, to match up the descriptor with the mounted filesystem. Our implementation is built on top of the `libfuse` library as depicted in Figure 2.

2.3 System Architecture

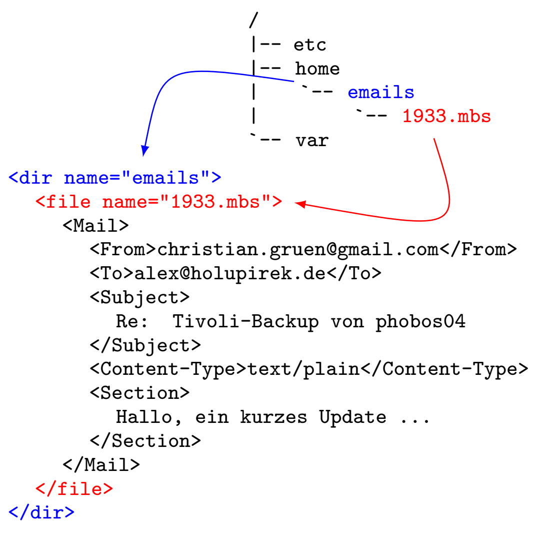
From a user's perspective, the system provides two access paths to the filesystem. Conventional/legacy access for any application can be achieved as exemplified by the `cat` command. The (virtual) filesystem operations relevant to print the file to standard output are looped back into userspace and captured by the functions registered with the callback interface of the `libfuse` library. The DEEPFUSE implementation[3] is responsible for translating the filesystem operations into corresponding XQuery/XQUF requests.

The MonetDB/XQuery server offers a well-defined and easy to use API (encapsulated in the Mapi library) to leverage its functionality. DEEPFUSE communicates as a client with a running server using a database driver and a textual protocol. The most frequently used functions called by the FUSE implementation have been encapsulated in an XQuery Module. This is to the best advantage as it allows MonetDB/XQuery to prepare a query plan for those functions, resulting in a much faster execution.

The user level filesystem implementation operates on a filesystem XML representation valid against a W3C XML Schema Definition. A DEEPFUSE XML instance is a (possibly empty) collection of files. Following the UNIX tradition there are block and character special, directory, fifo, symbolic link, socket and regular file types. Filesystem metadata, the `stat` information (access time, protection mode, file size ...) and any information relevant to operate a traditional filesystem is placed in the `http://www.deepfs.org/fs` namespace. This namespace encapsulates the information needed to operate the database as a filesystem in userspace. It is stored in a separate XML collection inside the MonetDB/XQuery system.

The second access path (as depicted with the `xls` command) goes along the conventional database interface. The filesystem data is stored in the database, and functions implemented in the XQuery module are ready to be used. However, this is not the crucial point. We expect applications that use XML as their storage format to query their data

[3]We call it DEEPFUSE, because operations on the file hierarchy do not necessarily end at the file level, but can continue on the file's inherent structure. The navigation along the file hierarchy in our implementation is basically the same as the navigation inside the XML files.



```
/
|-- etc
|-- home
|     `-- emails
|           `-- 1933.mbs
`-- var
```

```
<dir name="emails">
  <file name="1933.mbs">
    <Mail>
      <From>christian.gruen@gmail.com</From>
      <To>alex@holupirek.de</To>
      <Subject>
        Re:  Tivoli-Backup von phobos04
      </Subject>
      <Content-Type>text/plain</Content-Type>
      <Section>
        Hallo, ein kurzes Update ...
      </Section>
    </Mail>
  </file>
</dir>
```

pre	post	size	level	*kind*	*content*
0	12	12	0	fdir	`dir @name='emails'`
1	11	11	1	freg	`file @name='1933.mbs'`
2	10	10	2	elem	`Mail`
3	1	1	3	elem	`From`
4	0	0	4	text	*christian.gruen@gmail.com*
5	3	1	3	elem	`To`
6	2	0	4	text	*alex@holupirek.de*
7	5	1	3	elem	`Subject`
8	4	0	4	text	*Re: Tivoli-Backup von phobos04*
9	7	1	3	elem	`Content-Type`
10	6	0	4	text	*text/plain*
11	9	1	3	elem	`Section`
12	8	0	4	text	*Hallo, ein kurzes Update ...*

Figure 3: From file hierarchy to relational storage. Translators include file content to allow content and structure based queries as well as the deep access functionality while navigating along the file hierarchy.

for partial content or to only update "dirty" nodes instead of reading, processing and writing back complete XML files in their textual format, as this is the case when stored in filesystems.

3. THE DATABASE AS PERSONAL INFORMATION MANAGER

BaseX [3, 13] is an XML database, developed at the University of Konstanz. It is using a storage format influenced by and derived from the XPath accelerator numbering scheme [14]. From the beginning, the system was designed to offer visual access to the stored data. Together with the so-called "simple query mode" the system allows end-users to access its capabilities without having to explicitly use XQuery. That is why the system is suitable for putting our FS/DB approach to the proof against Desktop Search engines and other Personal Information Management (PIM) tools. With the, in the meantime, almost complete implementation of the XQuery Full-Text Candidate Recommendation [1] and a full-text index capable of fuzzy word matching, BaseX can fully exploit the inclusion of textual contents of files.

3.1 Mapping a File Hierachy into the Database

By choosing a directory, BaseX offers a feature to "shred" the file hierarchy to a DeepFuse XML instance and open it as a database. Apart from minor tweaks in the visualizations the instance is treated and processed as any other database instance. Basic user level commands (`ls, du, locate ...`) are implemented to operate on a DeepFuse XML representation of a filesystem. They mimic their counterparts on the operating system's shell.

As mentioned before, the XML representation breaks with the long tradition to consider a file as just a sequence of bytes. It unseals the black-box and lets the classical file hierarchy immerse into the files itself. The consideration of content and structure opens the door for query languages to operate on the data.

3.2 Searching Files using XQuery Full-Text

A central issue of the inclusion of textual contents into the XML representation of the filesystem is to allow the full range of XQuery retrieval features on the data.

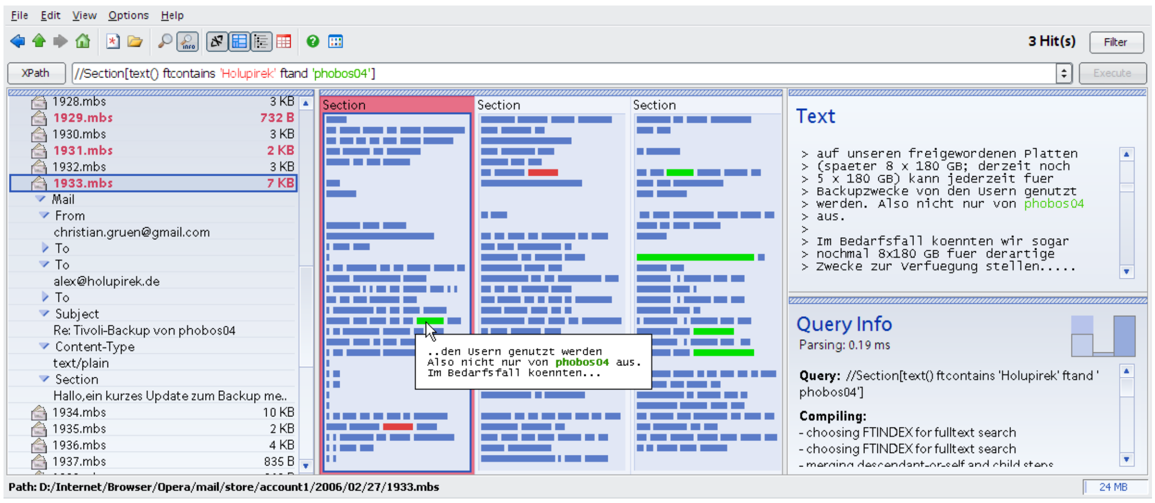
While XML files are ready to be included without additional effort (unless schema validation is demanded), commonly used binary files, such as images or audio files, contain metadata, which is quite relevant for querying.

BaseX provides a plug-in mechanism to easily develop and integrate translators for those file types. Several translators for various file types (*e.g., mp3, tif, png ...*) are already included. Textual files with inherent structure, such as e-mail, suggest themselves to be included with their structure exposed (see Figure 3). The current translator in BaseX produces a mapping, such as:

```
<file name="..." ...>
  <Mail>
    <Subject>...</Subject>
    <From>...</From>
    <To>...</To>
    <Content-Type>...</Content-Type>
    <Section>...</Section>
    <Attachment ...>...</Attachment>
  </Mail>
</file>
```

Although the implemented mappings are straightforward, they externalize formerly hidden information. The leverage of tacit information, formerly encapsulated in various formats, leads to a standardized and easily accessible representation. This provides a basis to operate on filesystem data with query languages. Think, for instance, about finding an e-mail (with a big attachment) on disk which was sent to you by a colleague in a specific mail:

```
for $mail in //Mail
let $attach := $mail/Attachment
where $mail/From = 'christian.gruen@gmail.com'
  and $mail/Section
    ftcontains 'Holupirek' ftand 'phobos04'
  and $attach/@size > 3000000
return deepfs:path($attach)
```



```
//Section[text() ftcontains 'Holupirek' ftand 'phobos04']
```

1928.mbs	3 KB
1929.mbs	732 B
1930.mbs	3 KB
1931.mbs	2 KB
1932.mbs	3 KB
1933.mbs	7 KB
Mail	
From	
christian.gruen@gmail.com	
To	
To	
alex@holupirek.de	
To	
Subject	
Re: Tivoli-Backup von phobos04	
Content-Type	
text/plain	
Section	
Hallo,ein kurzes Update zum Backup me..	
1934.mbs	10 KB
1935.mbs	2 KB
1936.mbs	4 KB
1937.mbs	835 B

```
..den Usern genutzt werden
Also nicht nur von phobos04 aus.
Im Bedarfsfall koennten...
```

```
> auf unseren freigewordenen Platten
> (spaeter 8 x 180 GB; derzeit noch
> 5 x 180 GB) kann jederzeit fuer
> Backupzwecke von den Usern genutzt
> werden. Also nicht nur von phobos04
> aus.
>
> Im Bedarfsfall koennten wir sogar
> nochmal 8x180 GB fuer derartige
> Zwecke zur Verfuegung stellen.....
```

Parsing: 0.19 ms

Query: //Section[text() ftcontains 'Holupirek' ftand ' phobos04']

Compiling:
- choosing FTINDEX for fulltext search
- choosing FTINDEX for fulltext search

Path: D:/Internet/Browser/Opera/mail/store/account1/2006/02/27/1933.mbs

Figure 4: BaseX highlighting the result matches of a full-text query. On the left, the continuation of the file hierarchy along the file's inherent structure can be seen.

Queries may combine filesystem metadata (such as file size, directory names) with file content and use both filesystem commands or database languages to query or manipulate the data. In the case of e-mail, comparable functionality is already offered by advanced e-mail applications. However, each application has to provide its own implementation leading to highly redundant code for similar functionality. Our approach strives to provide such capabilities as a basic service of the filesystem layer. Furthermore, the search is not restricted to application defined communication paths (such as the often connected e-mail, calendar, address book applications), but can include any data stored in the filesystem.

3.3 (Visual) Access to Filesystem Data

BaseX provides visual result presentation (see Figure 4 and 5). The central component is a space-filling visualization for hierarchic information, the treemap [23]. It makes extensive use of semantic zooming. In this context semantic zoom is to be looked upon as a form of details-on-demand technique which lets the user see different amounts of detail in a view by zooming in and out. The visualization is closely related with the description of data in the filesystem as "zooming in" correlates with "navigating into" a file.

When using desktop search engines users expect immediate responses on their keyword-based queries. BaseX answers those expectations by visually presenting the result set and allowing instant access on the data (*e.g.*, the images in Figure 5 or the actual e-mail in Figure 4—both can directly be accessed by double-clicking).

While we consider keyword-driven search a suitable approach for end-users and ad hoc queries, we assume that it may not be enough to cope with the explosive growth of personal information and the full variety of present and future user search tasks. With the tight coupling between different kind of query strategies (keyword-based, full-fledged XPath/XQuery) and result presentation, we try to establish a query cycle, which allows to refine, *i.e.*, filter/select/modify the (intermediate) results in a user-system feedback loop. This, for instance, allows to start a search with a simple keyword-based query, browse the result items and refine a selected context set by issuing an XQuery.

4. RELATED WORK

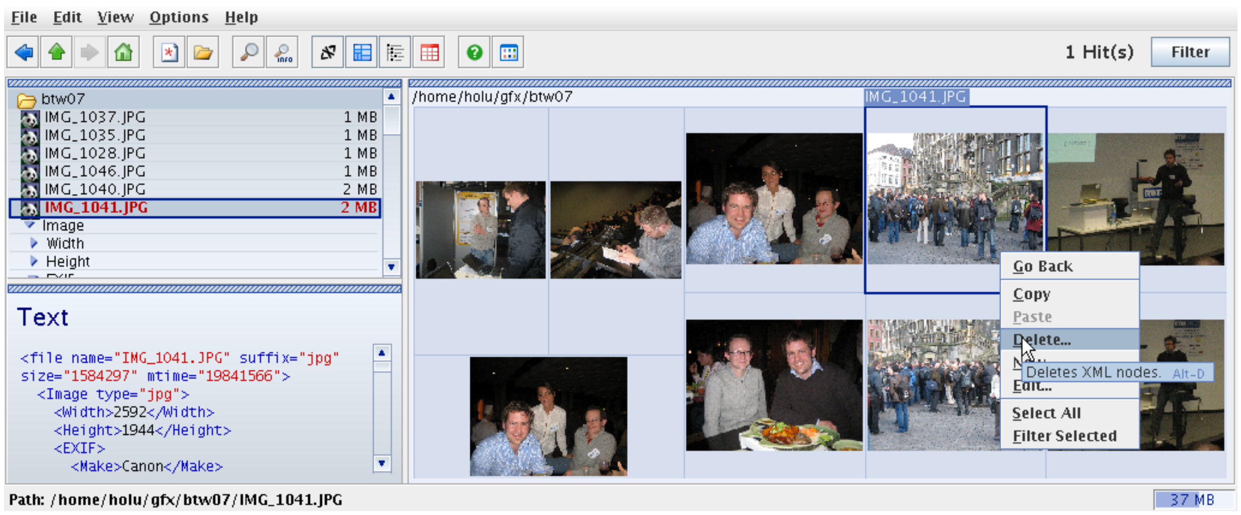
Various ideas have been proposed for including file contents into information systems.

One of the earliest attempts, the Semantic File System [10], extracted attribute-value pairs for specific file types via so-called transducers. Content queries could be formulated by entering directory paths and extending them with AND combined query terms. The result was a virtual path, resembling a default directory path and including symbolic links to the result documents. While SFS offered only limited retrieval functionality and ways of representing the query results, it has influenced numerous future filesystem projects, including Shore [8], HAC [11], or the recently dropped WinFS from Microsoft.

An interesting approach to bring XML and filesystems together was presented by IBM's XMLFS [2]. The underlying prototype implementation offered access to XML documents via an NFS server, and a simple path language allowed querying tags and text nodes across several documents. Nevertheless, the project was not extended to a full XPath/XQuery support, and document storage was apparently limited to XML instances and to the existence of DTDs.

The visionary paper [9] proposes dataspaces as a new data management abstraction. It led to various promising research efforts regarding the development of software platforms to facilitate a heterogeneous and distributed mix of personal information, such as Semex [7]. Approaches such as this are far more prospective and target the development of so called DataSpace Support Platforms (DSSPs). These are supposed to meet the criteria defined in [19]. In this context our work can be seen as a facet inside a DSSP.

IBM's Virtual XML Garden [27] and the draft of File System XML (FSX) [32] share the common idea to have a unified view over heterogeneous data sources. Since filesystems are structured hierarchically, they can easily be mapped to an XML structure as sketched in [32]. Together with the idea to let the filesystem immerse into the file [29], these provide the basis for the construction of our filesystem instance.

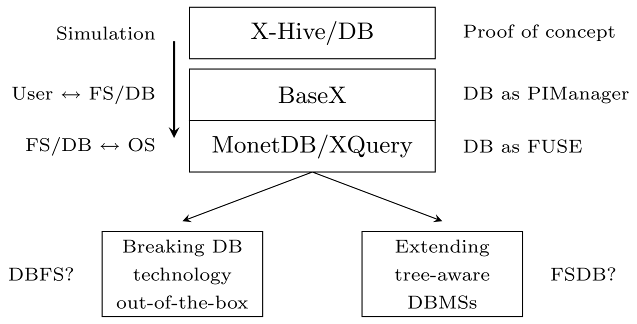


```
<file name="IMG_1041.JPG" suffix="jpg"
size="1584297" mtime="19841566">
  <Image type="jpg">
    <Width>2592</Width>
    <Height>1944</Height>
    <EXIF>
      <Make>Canon</Make>
```

Figure 5: BaseX provides visual access to query results. The user is able to browse and manipulate the results and to further refine the result set by issuing further keyword-based or full-fledged (X)Queries.

5. PROJECT STACK AND FUTURE WORK

As mentioned before, our long-term research is focused on the question: "Is it possible to combine database and operating system technology to query the filesystem?". We follow a top-down approach to evaluate the feasibility of our plan. Whenever research yields promising results on one of the top layers we try to push the concept down until it eventually integrates into the operating system.



Hence, the first step was to provide a proof of concept at the topmost layer [22]. A simulation evaluated the question whether a state-of-the-art XML-aware database management system (XHive/DB) is capable of processing filesystem operations as well as demanded query functionality on filesystem data represented as XML in interactive time.

The result triggered the second step which is explicitly focused on the evaluation of relational XQuery techniques as summarized in [31]. We agree with the idea to leverage proven relational database technology to operate semi-structured data. The resulting tree-awareness of RDBMSs is fundamental for our approach to implement filesystems using databases. Given that, the second project level is split and the two approaches have been described throughout this paper:

With BaseX, we try to connect user and filesystem/database hybrid. We will focus on performance evaluation and usability studies in comparison to desktop search engines and personal information utilities.

MonetDB/XQuery is the reference implementation for relational XQuery processing. As described, the system offers dedicated APIs at different layers of its architecture making it suitable to establish a link between operating system and DBMS. We will use it to evaluate the (performance) trade-offs we have to pay for a "queryable" filesystem. The evaluation will be a central issue of our future work. According to the top-down approach the results will determine if further integration to the operating system will make sense at all. Apart from that, two main directions can be considered:

"Breaking database technology out-of-the-box" is geared towards isolating the relevant techniques to implement a storage layer that suits both, filesystem and database demands. One can think of using the XPath accelerator numbering scheme as a basis for a filesystem implementation. Interlinking database node ids with filesystem vnodes may be an approach to further investigate. This direction would strive after a smooth integration to the operating system kernel, as such the label *DBFS* in the project stack. The broad idea would be a storage that is tuned for filesystem access, but able to serve as a storage layer to the database. A very early prototype rendered it possible to use the low-level interface of the FUSE API to implement a backing storage for BaseX. This finally could lead to an implementation of an in-kernel filesystem capable of interacting with a DBMS in userspace.

"Extending tree-aware DBMSs" on the other side would more closely follow the approach presented here, *i.e.*, to implement the filesystem in the DBMS. However a tighter integration, pushing several components into the DBMS itself, is quite likely. This direction would initially stick with the FUSE approach and accept the obvious performance penalty. However, one could think about a purely domain-specific filesystem, for instance, using it just for user data. The argumentation would be in line with using an encrypted filesystem. The administrator/user decides to pay (a yet to be determined price) for a more advanced, respectively otherwise missing, functionality.

6. SUMMARY

While filesystems provide an easy and well-understood interface to the data, they lack important and demanded features like the ability to query the data. Applications such as desktop search engines or personal information management tools often use a keyword-based search approach, which undeniably provides a user-friendly information discovery mechanism. There is no need to know the structure or format of the data, let alone a query language.

Ideally however, all retrieval strategies, *i.e.*, with no, partial or full knowledge about structure, format and content of the data, should be supported. Existing tools usually index plain text content and thus have started to break the file content out of its black box. That is to say, while traditional filesystems do not care about the content of a file, the supporting user level applications which provide the missing search and retrieval functionality do take it into account. A consistent further development is to provide means to expose the inherent file structure to allow content and structure queries.

RDBMSs are the right choice to query vast amounts of structured data. Data stored in the filesystem, however, is very heterogenous. With the rise of XML, the database community has been challenged by semi-structured data processing, enhancing their field of activity. Ongoing research efforts made RDBMSs, such as MonetDB/XQuery, capable of operating on tree-shaped data. As more and more applications save their data as XML, those files are ready to be directly saved into a DBMS. To go through with the concept, the file hierarchy tree along with its metadata is to be stored in the database in the same fashion.

We contribute implementations establishing a link between users and the filesystem/database hybrid as well as DBMS and OS. The DBMS is finally mounted as a filesystem by the OS and offers its data to arbitrary applications via the established (virtual) filesystem interface as well as through its database interface. As such we demonstrate the possibility of providing legacy filesystem access while storing the data in the database and have it ready to be queried.

7. ACKNOWLEDGMENT

This research is supported by the German Research Council (DFG) Research Training Group GK-1042 “Explorative Analysis and Visualization of Large Information Spaces”. We like to especially thank Christian Grün for lively discussions, assistance & comments and also the anonymous reviewers for providing valuable feedback.

8. REFERENCES

[1] S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, M. Holstege, J. Melton, M. Rys, and J. Shanmugasundaram. XQuery 1.0 and XPath 2.0 Full-Text. World Wide Web Consortium Candidate Recommendation, May 2008.

[2] A. Azagury, M. Factor, Y. S. Maarek, and B. Mandler. A Novel Navigation Paradigm for XML Repositories. *Journal of the American Society for Information Science and Technology (JASIST)*, 53(6):515–525, 2002.

[3] BaseX. Visual Exploration and Querying of XML Data. `http://www.basex.org/`.

[4] P. A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications.* Ph.D. Thesis, U Amsterdam, The Netherlands, May 2002.

[5] P. A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. Pathfinder: XQuery - The Relational Way. In *Proc. of the 31st Int'l Conference on Very Large Databases (VLDB)*, Trondheim, Norway, 2005.

[6] P. A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Chicago, Illinois, USA, June 2006.

[7] Y. Cai, X. L. Dong, A. Y. Halevy, J. M. Liu, and J. Madhavan. Personal Information Management with SEMEX. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Baltimore, Maryland, USA, June 2005.

[8] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, S. J. White, and M. J. Zwilling. Shoring Up Persistent Applications. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Minnesota, USA, May 1994.

[9] M. J. Franklin, A. Y. Halevy, and D. Maier. From Databases to Dataspaces: A New Abstraction for Information Management. *SIGMOD Record*, 34(4):27–33, 2005.

[10] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. O'Toole. Semantic File Systems. In *Proc. of the 13th ACM Symposium on Operating System Principles*, pages 16–25, California, USA, October 1991.

[11] B. Gopal and U. Manber. Integrating Content-Based Access Mechanisms with Hierarchical File Systems. In *Proc. of the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–278, February 1999.

[12] J. Gray, A. S. Szalay, A. Thakar, C. Stoughton, and J. vandenBerg. Online Scientific Data Curation, Publication, and Archiving. *CoRR*, 0208012, 2002.

[13] C. Grün. Pushing XML Main Memory Databases to their Limits. In *Proc. of the 18th GI-Workshop on the Foundations of Databases*, pages 60–64, June 2006.

[14] T. Grust. Accelerating XPath Location Steps. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Madison, Wisconsin, June 2002.

[15] T. Grust, M. Mayr, J. Rittinger, S. Sakr, and J. Teubner. A SQL:1999 Code Generator for the Pathfinder XQuery Compiler. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Beijing, China, June 2007.

[16] T. Grust, J. Rittinger, and J. Teubner. Why off-the-shelf RDBMSs are better at XPath than you might expect. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, Beijing, China, June 2007.

[17] T. Grust, S. Sakr, and J. Teubner. XQuery on SQL Hosts. In *(e)Proc. of the 30th Int'l Conference on Very Large Data Bases (VLDB)*, Toronto, Canada, August 2004.

[18] T. Grust and M. van Keulen. Tree Awareness for Relational DBMS Kernels: Staircase Join. In *Intelligent Search on XML Data*, volume 2818 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 2003.

[19] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of Dataspace Systems. In *Proc. of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Chicago, Illinois, Maryland, USA, June 2006.

[20] C. Henk. FreeBSD Port of the FUSE Framework. `http://fuse4bsd.creo.hu/`, 2007.

[21] D. Hiemstra, H. Rode, R. van Os, and J. Flokstra. PF/Tijah: Text Search in an XML Database System. In *Proc. of the 2nd International Workshop on Open Source Information Retrieval (OSIR)*, 2006.

[22] A. Holupirek, C. Grün, and M. H. Scholl. Melting Pot XML: Bringing Filesystems and Databases One Step Closer. In *Proc. of the 12th GI-conference on Database Systems in Business, Technology and Web (BTW)*, Aachen, Germany, March 2007.

[23] B. Johnson and B. Shneiderman. Tree Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures. In *Proc. of the 2nd International IEEE Visualization Conference*, pages 284–291. IEEE Computer Society, 1991.

[24] A. Kantee. puffs - Pass-to-Userspace Framework File System. In *Proc. of the 2nd Asia BSD Conference (AsiaBSDCon)*, 2007.

[25] A. Kantee and A. Crooks. ReFUSE: Userspace FUSE Reimplementation Using puffs. In *Proc. of the 6th European BSD Conference (EuroBSDCon)*, 2007.

[26] D. A. Norman. *Emotional Design: Why We Love (Or Hate) Everyday Things*. Basic Books, January 2004.

[27] K. H. Rose, S. Malaika, and R. J. Schloss. Virtual XML: A Toolbox and Use Cases for the XML World View. *IBM Systems Journal*, 45(2):411–424, 2006.

[28] A. Singh. A FUSE-Compliant File System Implementation Mechanism for Mac OS X. `http://code.google.com/p/macfuse/`.

[29] S. St.Laurent. Bringing the File System into the File. `http://www.simonstl.com/articles`, 1998.

[30] M. Szeredi. Filesystem in USErspace. `http://fuse.sourceforge.net/`.

[31] J. Teubner. *Pathfinder: XQuery Compilation Techniques for Relational Database Targets*. Ph.D. Thesis, TU München, Germany, Oct 2006.

[32] E. Wilde. Merging Trees: File System and Content Integration. In *Proc. of the 15th Int'l Conference on World Wide Web*, pages 955–956, Edinburgh, Scotland, UK, May 2006.

[33] E. Zadok and J. Nieh. FiST: A Language for Stackable File Systems. In *Proc. of the USENIX Annual Technical Conference, General Track*, pages 55–70, San Diego, CA, USA, June 2000.