Files are Siles: Extending File Systems with Semantic Annotations*

Bernhard Schandl and Bernhard Haslhofer

University of Vienna, Department of Distributed and Multimedia Systems
{bernhard.schandl,bernhard.haslhofer}@univie.ac.at

Abstract. With the increasing storage capacity of personal computing devices, the problems of information overload and information fragmentation become apparent on users' desktops. For the Web, semantic technologies aim at solving this problem by adding a machine-interpretable information layer on top of existing resources. It has been shown that the application of these technologies to desktop environments is helpful for end users. Certain characteristics of the Semantic Web architecture that are commonly accepted in the Web context, however, are not desirable for desktops; e.g., incomplete information, broken links, or disruption of content and annotations. To overcome these limitations we propose in this paper the sile model, an intermediate data model that combines characteristics of the Semantic Web and file systems. This model is intended to be a conceptual foundation of the Semantic Desktop, and to serve as underlying infrastructure on which applications and further services, can be built. We present one such service, namely a virtual file system based on siles, which allows users to semantically annotate files and directories but at the same time keeps full compatibility to traditional hierarchical file systems; hence, users can continue to use file-based applications. We discuss strategies how Semantic Web vocabularies can be applied for meaningful annotation of files. Further, we present a prototypical implementation of our model and analyze the performance of typical access operations, both on the file system level as well as on the metadata level.

1 Introduction

Large amounts of information are stored on personal desktops. We use our personal computing devices—both mobile and stationary—to communicate, to write documents, to organize multimedia content, to search for and retrieve information, and much more. With the increasing computing and storage power of such devices, we face the problem of *information overload*: the amount of data we generate and consume is permanently increasing, and because of the availability of cheap storage space, each and every bit of information is stored. Another problem is even more prevalent on the desktop than on the Web: *information fragmentation.* Data of different kinds are stored in heterogeneous silos, and—contrary

* This paper is an extended version of [31].

to the Web, where hyperlinks can be defined between documents and across site boundaries—there exist only limited means to define and retrieve relationships between different desktop resources. In the best case such relationships can be represented using additional infrastructure (e.g., relational databases or specific applications), but these are usually not tightly integrated with file systems.

The Semantic Web aims to deal with the problems mentioned before by adding a layer on top of the existing Web infrastructure, wherein descriptions about web resources are expressed using the Resource Description Framework (RDF) using commonly accepted vocabularies or ontologies. This allows machines to interpret the published data and thus helps end users to find information more efficiently. A large number of data sets[1] and vocabularies[2] have already been published and form a solid data corpus that can be indexed by (semantic) search engines and serves as foundation for applications.

Recent research in the field of the Semantic Desktop [6, 17, 20] has shown that a number of features provided by Semantic Web technologies are also suitable for the problem of information management on the desktop; especially, the provision of unified identifiers, the ability to represent data in an application-independent generic format, the flexibility to describe resources using formalized vocabularies, and the possibility to reason over these descriptions. It has also been shown [28, 13] that the inclusion of semantic technologies on the desktop can significantly improve the user's perceived quality of personal information management, especially when they are applied during a longer time period.

However there exist some significant conceptual differences between the Web and the desktop. First, in contrast to the World Wide Web, the desktop already has a well-established organization metaphor for data: *file systems*, which have been in use for decades. In consequence, the vast majority of personal information are stored in files, which are organized using hierarchical, labelled collections (*folders* or *directories*) or, to a far more limited extent, using metadata attached to or encoded within files. Therefore it is crucial for the Semantic Desktop to smoothly integrate with file systems in a way that allows for the annotation of files without breaking the behavior of existing desktop applications. A second major difference is the handling of broken links. While appearing and disappearing web resources are—to a certain extent—accepted on the Web, users rightfully expect their data on the desktop to remain consistent over time.

Since the RDF data model exposes a number of shortcomings that may cause problems for an efficient implementation of the Semantic Desktop (cf. Section 3), we propose the *sile model*, a data model that acts as an intermediate and integrative layer between file systems and Semantic Web technologies. This model allows users and applications to annotate and interrelate file-like desktop resources. It is designed as an infrastructure on which applications and services can be built. One example of such a service, a virtual file system, is presented in this paper. Through this virtual file representation, the sile model can be used

[1] http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets
[2] http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/CommonVocabularies

as a hierarchical file system and thus maintains full backwards-compatibility to existing systems and applications.

The sile model has been designed to be interoperable with vocabularies from the Semantic Web in order to establish a unified, homogeneous data space that encompasses the desktop and the Web. It enables tools and applications to operate on data both locally and globally, and data can be seamlessly exchanged between these two worlds. Hence we propose to use Web vocabularies for data representation wherever possible, and we support this requirement by using URIs—which may refer to Semantic Web resources—as identifiers for all annotations in our data model.

After an analysis on related work in the field of semantic file systems and Semantic Desktop technologies (Section 2), this paper discusses the issues that arise when file systems and semantic technologies are integrated (Section 3) and presents the sile model (Section 4) as a data abstraction layer for the desktop. We discuss how the sile model can be used in applications, and propose a set of already existing Semantic Web vocabularies for the annotation of desktop data (Section 5). Finally, we present our prototypical implementation (Section 6), which we have evaluated under the assumption of realistic amounts of data (Section 7).

2 Related Work

For a long time, file systems have been an interesting application field for metadata. The drawbacks and deficiencies of hierarchical file systems have been identified and described in various works [7, 32, 35, 38], which has lead to research and industry effort that aimed at integrating hierarchical file systems with metadata and semantic annotations. In this respect, we can observe two development lines: on the one hand, hierarchical file systems that can be found as part of common operating systems have been extended with support for file annotations and descriptions. On the other hand, a new class of file systems have been developed, which reduce the prominence of directory hierarchies in favor of a more metadata-centric approach. These systems often do not rely on a physically existing hierarchy, but instead virtually emulate it based on file annotations.

Extensions to Hierarchical File Systems. Originally, hierarchical file systems did not provide mechanisms to attach user- or application-defined metadata to files. Directory and file names were the only existing organization metaphor, and at most a limited pre-defined set of metadata attributes (such as date of creation and last update) was maintained. However, it has been recognized that a strict tree-based structure is not always appropriate for file management, and hence several mechanisms were introduced that allow files to appear in more than one directory, including *hard links* or *symbolic links*. While links solve a number of file categorization problems, they do not allow to further describe file characteristics in a structured manner, or to bilaterally relate files.

Modern file systems like *NTFS*, *HFS*, or *ext3* provide—in addition to directory hierarchies and links—support for different kinds of metadata. We can

	Hard / Symbolic Links	Tags / Keywords	Attributes	Orthogonal Categories	Internal Relationships	External Relationships	Multiple Streams / Forks	Full Read and Write Support	Virtual File Hierarchy	API	Structured Query Language
FAT								✓	n/a		
NTFS	✓		✓				✓	✓	n/a	✓	
HFS	✓		✓				✓	✓	n/a	✓	
ext3 / ext4	✓		✓					✓	n/a	✓	
WinFS		✓	✓	✓	✓			✓	✓	✓	✓
SFS	✓		✓						✓		✓
AttrFS	✓	✓	✓					✓	✓		✓
Presto			✓			✓			✓	✓	✓
LISFS			✓	✓					✓		✓
TagFS	✓	✓	✓					✓	✓	✓	✓
libferris			✓					✓	✓	✓	✓
LiFS	✓		✓		✓		✓	✓	✓	✓	✓
SileFS	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓

Fig. 1: Comparison of metadata support and access mechanisms in hierarchical file systems and metadata-centric file systems

roughly distinguish between systems that provide *file forks*, which allow for storing additional content in separate data areas alongside the file content, and *file attributes*, which provide the means to attach structured name/value pairs to files. Figure 1 gives an overview of modern file systems and the annotation features they support[3]. While file forks and file attributes provide sufficient means to attach meaningful metadata to files, they are still not expressive enough to express relationships between files in a stable and consistent manner, or to assign predefined types (classes, categories) to files. Works in this area are rooted in

[3] Note that the actual naming of file system features differs; for instance, file forks are called *alternate data streams* in NTFS, while file attributes are called *extended attributes* in ext3/ext4.

research on object-oriented data bases (e.g., [9]) and would have been integrated into WinFS [16] whose development however halted in 2006.

Metadata-centric (Virtual) File Systems. Despite the support for file metadata provided by modern hierarchical file systems, hierarchical directory structures are still accentuated as the primary mechanism for file organization. A number of alternative approaches were proposed that raise the role of file metadata as the main file access method. Figure 1 gives an overview on their most important features.

The Semantic File System [15] presented by Gifford uses structured name/value pairs, which are extracted from analyzing file contents, to establish a virtual directory hierarchy. The elements of a directory path represent a conjunction of search criteria. Gifford's system is, however, read-only since attribute values are derived only from extracting file features; support for write operations was introduced later in AttrFS [38]. Presto [12] follows a similar path by providing a virtual hierarchical view on annotated files. LISFS [25] extends the attribute-based virtual file system approach by interpreting paths as Boolean formulas over the file space. TagFS [5], which is built on a generic framework for semantic file systems, uses tags (keywords) instead of structured name/value pairs to simulate a directory hierarchy. libferris [21] focuses on the integration of different resources (like XML documents, plain files, or running applications) and represents them as a single unified virtual hierarchical file system; additional user-defined annotations are stored in a designated RDF store that is embedded in the platform. Finally, LiFS [1] represents relationships between files as virtual directories alongside the actual files, an approach also followed by LODFS [30] where remote Linked Open Data sources (i.e., plain RDF graphs) are represented as virtual file systems.

The main problem with approaches that represent file metadata in the form of virtual directories is that they break some core characteristics of hierarchical file systems; however many file-based applications and services are implemented under the assumption of these characteristics. For instance, in a hierarchical file system an application can safely assume that a file is stored at exactly one location, and that different paths refer to different files (with the exception of symbolic or hard links): access in a hierarchical file system is based on *location*, not on *description*. If this assumption does not hold (as it is in the case of metadata-centric virtual file systems), software that crawls a file system (e.g., search engines) may run into problems because they are not able to determine whether two files are actually the same.

Moreover, virtual directory hierarchies do not provide *stable references* to external systems. In hierarchical file systems, the primary mechanism to refer to a file is its full path. It is clear for users and applications that file references become invalid if a file is moved or renamed. However, if the full path of a file contains metadata attributes (which, for instance, may be derived from the file contents or may reflect the user's opinion, which may change over time) these references become invalid when the file's metadata change; essentially this means that a modification of a file's content may cause its references to break.

Another problem with metadata-based virtual directory hierarchies is that in many cases the semantics of write operations is undefined. Consider, for instance, a virtual directory `/(proj1,proj2,proj3)` that contains files that are tagged with `proj1`, `proj2`, or `proj3`, or a combination thereof [38]. While such virtual directories are helpful for search and retrieval, it is unclear how a file that is written into such a directory should be annotated. This restriction significantly reduces the benefit, convenience, and practical applicability of purely annotation-based virtual file systems.

Finally, it is unclear what the actual target audience and practical use cases of metadata-centric virtual file systems are. For applications that are based on hierarchical file systems, metadata-centric virtual directories are opaque since they do not understand the annotations' semantics, and therefore bring no additional benefit. For metadata-aware applications, the employment of a dedicated annotation model and an associated structured query language seems to be more practical from the authors' perspective. For end users, most approaches assume the usage of a command line to enter paths that represent search queries (usually via the `cd` and `ls/dir` commands). We doubt however that the typical end user will be sufficiently familiar with command line interfaces in order to efficiently work with this method: it requires one to memorize not only the required commands, but also the exact names of tags, attributes, and other annotations. Graphical, icon-based interfaces can be used in combination with virtual file systems although these distinguish only between directories and files and are not aware of semantic file annotations. The formulation of more complex queries (as presented before) through graphical interfaces is often not possible at all; consequently the efficiency of navigation and browsing is reduced.

Semantic Desktop. The idea of the Semantic Desktop goes one major step further than semantic file systems do. The idea of this approach is not only to enrich file systems with expressive annotations, but also to integrate the entire information sphere of a user into a semantic network [10, 27], which has the potential to significantly increase the quality of information work [13]. This is accomplished either by providing services that maintain a layer of semantic descriptions that refer to actual resources like files, e-mails, and web pages, while users continue to work with their well-known applications (e.g., Nepomuk [17], Semex [8], or iMeMex [6]), or by providing an integrated work environment that disbands the traditional application-centric paradigm of desktop computers in favor of a data- and annotation-centric work style (e.g., Haystack [20], DeepaMehta [26], or X-COSIM [14]). As such, the Semantic Desktop can be considered as an orthogonal architecture that may embrace all kinds of storage mechanisms, including traditional as well as semantic file systems.

One significant outcome of Semantic Desktop research is the integration of semantic technologies into the *K Desktop Environment*[4] (KDE). An RDF database is now part of its standard distribution and enables the system-wide storage and retrieval of semantic annotations for files and other information items. References between the physical storage systems and the annotation store are maintained by

[4] Semantic Desktop with KDE: `http://nepomuk.kde.org`

using kernel notification services. Annotated files can be accessed transparently through virtual folders, which are supported by the system-provided dialogs for opening and saving files. Although Nepomuk-KDE is not a semantic file system in the narrower sense, its increasing adoption by applications and users shows that there exists a significant need for more expressive file annotations.

Approaches in the field of the Semantic Desktop go far beyond the functionality and envisioned use cases of a metadata-enriched file system. However, semantically enriched file systems are highly complementary to semantic desktop approaches since they provide an integrated storage infrastructure for files and annotations without the need for additional synchronization mechanisms.

3 Integrating File Systems and RDF Descriptions

File Systems as a Data Organization Metaphor. The majority of personal information is stored within file systems, where we can observe three usage patterns. We denote the first one as *user-driven file structures*: a large number of applications do not use internal directory structures or file name conventions, but allow users to organize files in directories named and nested according to their subjective needs. Examples for this group of applications include word processors, spreadsheet tools, or graphics suites. A second group establishes *application-driven file structures*, where directory hierarchies and files are managed entirely by applications, but still expose a certain meaning to end users. Examples of such applications are e-mail clients, which store email messages as files in a directory hierarchy and allow the user to manipulate these hierarchies through a dedicated interface, or media players, where the user has no influence on the handling of directories; here, the application uses file metadata (e.g., artist name or song title) as directory names. Typically, these applications do not operate on single files but on file collections, and interpret the directory hierarchies and file names in a semi-schematic manner. This pattern is also used for application-internal files that do not store user data; however the application requires certain files to be found in certain locations in order to work correctly. A third group of applications do not expose the semantics of file structures to the user, but rather operate on continuous data corpora; examples for this group include calendar applications that store appointments in one large file, or database-driven applications. We denote the data structures of such applications as *hidden file structures*, because in the end even these data are stored in file systems, although not semantically organized by the file system's means. Figure 2 summarizes characteristics of these three groups of structures w.r.t. their application- and user-related semantics.

For all three classes of applications it is desirable to make information accessible in a semantically meaningful, application-independent way. This has two main benefits: first, it makes it possible to interrelate objects from different sources, which otherwise form disjunct structures in one's personal information space [7]; and second, it extends the possibilities provided by file systems with respect to search and retrieval [28]. All three classes of applications, as described

Characteristics #	**User-driven**	**Application-driven**	**Hidden**
Application-defined semantics		✓	✓
Application-interpretable semantics		✓	✓
User-defined semantics	✓		
User-interpretable semantics	✓	✓	
Readable by applications	✓	✓	✓
Modifiable by applications		✓	✓
Browsable by users	✓	✓	
Browsable by users	✓		

Fig. 2: Characteristics of user-driven, application-driven, and hidden file structures w.r.t. file and directory semantics

before, could benefit from such an integration: user-driven file structures could be extended by more powerful and non-hierarchical annotation mechanisms that are orthogonal to the physical identification of files, like tags and typed relationships. Application-driven file structures could be disbanded in favor of explicit semantic annotations that ideally adhere to open vocabularies and therefore are interpretable by external entities. Finally, the internals of hidden file structures could be revealed and hence be integrated with other information sources.

The Gap to RDF. The RDF data model can be used to identify information units (*resources*) and to describe these units and the relationships between them using subject-predicate-object triples. Its successful usage in the context of Linked Open Data [3] indicates that it is sufficiently expressive for a large number of applications, ranging from statistical data via geo information to descriptions of multimedia content. Certain RDF characteristics, however, are problematic in the context of the desktop. First, RDF does not consider the actual *content* of information items. The relationship between a resource identifier (a URI) and the resource itself has been explicitly left out of the scope of the RDF specification, which states, "[...] *no assumptions are made here about the nature of resources; 'resource' is treated here as synonymous with 'entity', i.e. as a generic term for anything in the universe of discourse*" (cf. [18], Section 1.2). In the open Web environment, broken links and unavailable resources are likely to occur; solutions to maintain data quality are still an open research issue [4]. However, on the desktop, applications must be enabled to rely on a consistent view on content objects and their annotations at any time, hence this integrity must always be ensured.

Another potentially problematic aspect of RDF is the Open World Assumption. Since RDF is designed for the Web, incomplete or unknown information is accepted by design. Again, for the closed environment of the desktop, incomplete information is not desirable; here, the Open World Assumption hinders the efficient realization of certain features, e.g., negated queries. Consequently, recent approaches to Semantic Desktop information modeling (e.g., [36]) have restricted the interpretation of RDF data to a closed world semantics.

As a third problematic issue, the application of certain RDF language elements (in particular collections, blank nodes, and reification) is being discouraged both in the context of the Web (cf. [3], Section 2.2) and the desktop (cf. [37], Section 2.3.2). These features significantly increase the complexity of RDF processing, as well as representing RDF in user interfaces [22, 34], but expose a very light semantics [22]. Hence it is doubtful whether these elements are useful in the context of the desktop, where end users have only limited knowledge about the characteristics of the underlying data structures, and the available computing power is limited in comparison to large-scale database servers.

Recent Semantic Desktop projects (e.g., [6, 17, 20]) add an RDF layer on top of existing desktop infrastructures and refer to the actual content representation using local URIs. These URIs are often minted based on the characteristics of the underlying system. A URI that refers to a file on the user's hard drive, for instance, could be `file:///data/semdav/semdav_description.doc`. Such URIs, however, are not suitable as permanent identifiers for files [33]: first, they are *unstable* since they become invalid when files are renamed, moved to different directories, or deleted. Consequently, synchronization mechanisms are needed that propagate modifications to the semantic layer; however often it is difficult to track such modifications accurately. In the worst case, when such propagation is not possible, references to files become broken. Furthermore, path-based URIs are not *globally valid*, as they can be resolved only relative to the local system. This aspect is problematic when files are exchanged across machines, since the URIs may be no longer valid, or different files with equal URIs may occur, causing a name clash.

In the following, we present an alternative modeling approach that aims to solve the problems described before: we consider an item's content and its annotations as integral components, which are always processed together. Instead of adding a semantic layer on top of existing file structures and relying on unstable identifiers based thereon, we inject semantic annotations and globally valid, permanent identifiers into the core of the data representation structures. On top of this model a virtual, file system-like representation of the data stored therein is presented, which can be seamlessly accessed by existing tools and applications.

4 The Sile Model

Siles: Adding Semantics to Files. *Siles* (*S*emantic f*iles*) can be regarded as combinations of files and semantic annotations. A sile is always identified by a globally unique URI and consists of a (binary) string of arbitrary *content*, as well as an arbitrary number of *annotations*. In the context of siles, URIs are not used as URLs: while in the Semantic Web it is recommended that the URI of a resource is at the same time a URL that can be dereferenced in order to retrieve the resource's representation (cf. Figure 3a), this is not necessarily the case for siles. In our model, URIs are solely used for identification purposes, and the sile identifier, the content, and the associated annotations are integral parts of the sile, as depicted in Figure 3b.

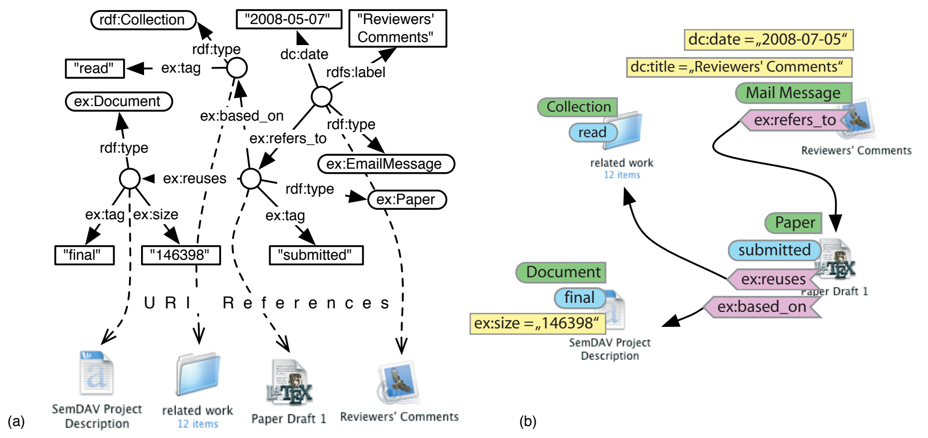

Fig. 3: (a) RDF model: URIs refer to actual content; (b) Sile model: integrated view on content and annotations.

The sile model distinguishes between four types of annotations, which are able to cover a large share of annotation needs in the desktop domain: *tags*, which are plain strings; *categories*, which refer to entities with machine-processable semantic interpretation (e.g., classes from an controlled vocabulary); *attributes* in the form of typed name/value pairs; and *slinks* (semantic links), i.e., directed typed relationships between siles. The names of categories, attributes, and slinks are URIs, which allow for an unambiguous interpretation of their semantics; however the formalism used for this purpose is out of the scope of the sile model. A category annotation, for instance, may refer to an OWL ontology class as well as to a table within a relational database schema[5]. Figure 3(b) depicts a number of siles and their associated annotations, whereas different shapes and colors are used to indicate different annotation types.

Formal Model. Let Σ denote the set of all siles in the university of discourse. Let $\mathbb{LIT}$ denote the set of all *string literals* which are finite sequences of characters from an *literal alphabet* α, and $\mathbb{B}$ the set of all *content literals* which are finite sequences of characters from an *content alphabet* β. Further, let $\mathbb{URI}$ denote the

[5] In Section 5 we outline a number of vocabularies that can be applied in desktop contexts.

set of all *Uniform Resource Identifiers (URIs)*. Let $\mathbb{T}$ denote the set of all *tags*, $\mathbb{T} \subseteq \mathbb{LIT}$, Let $\mathbb{C}$ denote the set of all *categories*, $\mathbb{C} \subseteq \mathbb{URI}$, and let $\mathbb{A}$ denote the set of all *attributes*, $\mathbb{A} = \mathbb{URI} \times \mathbb{LIT} \times \mathbb{URI}$. and let $\mathbb{L}$ denote the set of all *slinks*, $\mathbb{L} = \mathbb{URI} \times \mathbb{URI}$. Let $\mathbb{ANN}$ denote the set of *annotations*, $\mathbb{ANN} = \mathbb{T} \cup \mathbb{A} \cup \mathbb{C} \cup \mathbb{L}$.

Using this vocabulary we can define a sile $s \in \Sigma$ as a six-tuple $s = (u_s, b_s, T_s, C_s, A_s, L_s)$. $u_s \in \mathbb{URI}$ denotes the *URI* that uniquely identifies sile s, $b_s \in \mathbb{B} \cup \perp$ denotes the sile's *binary content*, $T_s \subseteq \mathbb{T}$ is the set of the sile's associated *tags*, $C_s \subseteq \mathbb{C}$ is the set of the sile's associated *categories*, $A_s \subseteq \mathbb{A}$ the set of the sile's associated *attributes*, and $L_s \subseteq \mathbb{L}$ is the set of the sile's associated *slinks*.

Within the sile model, we do not impose constraints on the structure or the nature of a sile's binary content. The model neither defines rules that state how the content is to be interpreted, nor how annotations can be deduced by analyzing sile content.

Based on their annotations, siles can be accessed using operators from an abstract query algebra [29]. This algebra provides *annotation existence predicates*, which indicate whether a given sile carries a specified annotation (e.g., $\mathrm{existsTag}_{\Gamma}(s, t)$, which returns whether sile s is annotated with tag t), as well as *sile selection operators*, which return sets of siles that fulfill certain criteria (e.g., $\mathrm{RelatedSiles}_{\Gamma}(s)$, which returns all siles that are related (linked) to sile s). All sile algebra operators are evaluated against a specified context Γ, e.g., a specific file system.

$$
\begin{aligned}
s_1 = (&\texttt{<urn:uuid:57207370-6880-11dd-ad8b-0800200c9a66>},\\
&\texttt{"9j4AAQSkZJRgABAQEAYABgAAAoHBwgHBgoICAgLCgoLEYIx8l...",}\\
&\{\texttt{"final"}, \texttt{"data"}, \texttt{"semdav"}\}, \{\texttt{silefs:File}\},\\
&\{(\texttt{sile:creation-date}, \texttt{"2008-07-11T16:21:14"}, \texttt{xsd:dateTime}),\\
&(\texttt{sile:update-date}, \texttt{"2008-07-11T17:32:02"}, \texttt{xsd:dateTime}),\\
&(\texttt{sile:content-type}, \texttt{"application/msword"}, \texttt{xsd:string}),\\
&(\texttt{sile:content-length}, \texttt{"146398"}, \texttt{xsd:long}),\\
&(\texttt{silefs:path}, \texttt{"/data/semdav/semdav\_description.doc"}, \texttt{xsd:string})\},\\
&\{(\texttt{dcterms:subject}, \texttt{<http://www.semdav.org>}),\\
&(\texttt{silefs:parent}, \texttt{<urn:uuid:60ad6a73-1b60-4553-9436-d09d395fc29c>})\} \quad)
\end{aligned}
$$

Fig. 4: A file represented using the sile model

The sile model resembles characteristics from other data models, in particular from file systems and from the Semantic Web technology stack. Similar to files, it provides the concept of self-contained information units that carry content of unlimited length and undefined inner structure, but at the same time it avoids the problems of unstable references and the blending of identification and annotation by providing stable, globally unique identifiers for these information units. The Sile model borrows from RDF the means to describe these information units using URI-based vocabularies, but it makes the semantics of these annotations more explicit by providing higher-level annotation concepts (tags, categories, attributes, and slinks) instead of plain triples. Likewise it avoids aspects of RDF that are less frequently used, like blank nodes or reification, in favor of a simplified and more streamlined model.

Representing Directory Hierarchies. A basic file system can be represented without loss of information by an unordered tree having two types of nodes: directories and files. Directory nodes may have children, whereas file nodes are always leaf nodes. We map each node in the directory tree to a sile and represent its type by a category annotation (`silefs:File` or `silefs:Directory` $\in C_s$, respectively). We reference an object's parent directory using a slink that refers to the parent object's URI (e.g., $(\texttt{silefs:parent}, U_{parent}) \in L_s$).

By concatenating the labels of the nodes along a path, a unique identifier for each element (directory or file) of the file system can be constructed. However, when a path expression consists of many elements, traversal across a large fraction of the graph is required in order to locate the described file. To enable direct access to a file given its path, we additionally store the file's absolute path as a sile attribute (`silefs:path`, `"/data/semdav/semdav_description.doc"`, `xsd:string`) $\in A_s$). This leads to processing overhead when the file tree is modified, but significantly increases the performance of read operations. Figure 4 depicts a sile in its abstract notation[6]. Here, the explicitly saved file path is also reflected by automatically generated tags, which are updated transparently when a file is created or moved.

Operations that involve a directory and its children (e.g., a directory listing) can be resolved by querying for `silefs:parent` relationships between siles. When a file or directory is renamed or moved[7], the steps to be taken depend on whether the file remains within its parent directory or not. In the former case, it is sufficient to update the sile's `silefs:path` attribute, while in the latter case the `silefs:parent` slink must additionally be updated to refer to the sile's new parent directory. However, the identity of this sile can be efficiently retrieved by querying for the directory sile's `silefs:path` attribute.

The representation of files using a graph-based model also enables us to represent hard or symbolic links by storing additional `silefs:path` attributes and `silefs:parent` relationships. The semantics of the deletion of a link can be simulated by checking whether a sile has more than one `silefs:path` attribute before deletion, and by only deleting the entire sile if it has only one such attribute. In the case of multiple `silefs:path` attributes, only the attribute and the `silefs:parent` slink are deleted.

We like to stress the fact that, contrary to approaches that derive virtual directories from semantic file metadata (usually, tags or attributes), our approach

[6] For the sake of readability we use CURIE notation [2] to abbreviate URIs.

[7] Moving and renaming of files are implemented identically in many operating systems.

uses *designated annotations* with well-defined names and semantics in order to explicitly represent a directory structure. Additional annotations can be derived from this information (e.g., the tags depicted in Figure 4). By doing so the danger of accidentally breaking file path references by changing file annotations is avoided. While our system in principle allows for metadata-based virtual directories, it would again lead to the undesired effects that common semantic file systems suffer from, as described in Section 2.

5 Annotating Files Using the Sile Model

Accessing Semantic Files. Using the mechanisms described in the previous section, we can now implement a common hierarchical file system that can be directly used by existing file-based applications. To read, write, and search for semantic annotations of files that are represented as siles, we have developed an Application Programming Interface (API) that is based on the formal specification of the sile data model as well as its associated query algebra (cf. Section 4). This API defines a type hierarchy for siles and annotations as well as corresponding access methods. To give the reader an idea, Figure 5 shows an excerpt of the class hierarchy for annotations, and Figure 6 shows a code example, which reproduces the annotations depicted in Figure 3b: a category and a tag are attached to a file. Later, the application can easily retrieve the object by searching for these annotations by using a conjunctive filter, as shown in Figure 7.

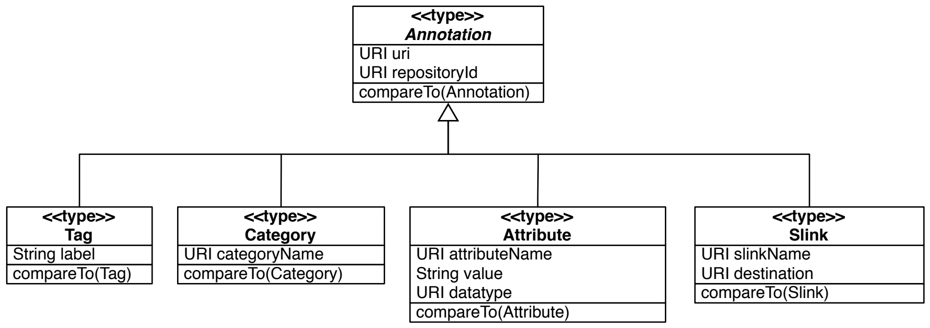

Fig. 5: Sile API: annotation types

The goal of this API is to allow desktop application developers to easily integrate semantic annotations into their code. Assuming that files are stored on a sile-based virtual file system, one can retrieve the sile that represents a certain file by a single API call, and access or manipulate this sile's annotations. Additionally, the API allows an application to retrieve siles that match certain criteria and to determine the corresponding file paths. A word processing application, for instance, could be extended so that it stores metadata about created documents (e.g., author and title) as sile attributes, or a search operation could be used to retrieve files that are associated with a certain project.

```
// write data for one file
File docFile = new File("/data/semdav/semdav_description.doc");
docFile.write(...);
Sile docSile = SileFS.getSileForFile(docFile);
docSile.addAnnotation(SileFS.getCategory("nfo:PaginatedTextDocument"));
docSile.addAnnotation(SileFS.getTag("final"));

// write data for second file
File paperFile = new File("/data/papers/ijswis/paper.tex");
paperFile.write(...);
Sile paperSile = SileFS.getSileForFile(paperFile);
paperSile.addAnnotation(SileFS.getSlink("ex:based-on", docSile));
```

Fig. 6: Sile API: write operations

```
// search all files that are tagged with "final" and linked to paperSile
Filter f1 = new TagFilter("final");
Filter f2 = new SileSlinkFilter(paperSile);
File[] results = SileFS.searchSiles(new AndFilter(f1, f2)).asFiles();
```

Fig. 7: Sile API: search operations

We believe that our API, which tightly integrates files and semantic annotations, has the potential to significantly increase the proliferation of semantic technologies on the desktop. As described in Section 3, the majority of desktop applications operate directly on the file system. By transferring existing file hierarchies to a virtual, sile-based file system and by integrating semantic annotations into applications, the desktop can be extended using semantic technology while avoiding the need for fundamental architecture changes for single applications or entire operating systems.

Deriving Semantic Annotations for Files. Since siles are exposed as a virtual file system that emulates the behavior of a common hierarchical file system, they can be accessed by legacy applications and tools (e.g., file browsers) without changing any code. Other virtual file system views based on sile annotations (e.g., tags) can be additionally instantiated and therefore exposed to applications, under consideration of the problems imposed by this approach (cf. Section 2). Structured annotations can be extracted automatically from any file by applying content- or interaction-based feature extraction (e.g., [23]). Only applications that explicitly access file annotations and therefore follow their semantics, which may be defined e.g., using a formal schema, must be adapted to use the sile API.

We have shown in the previous section that the API for creating and retrieving sile annotations is easy to learn and straightforward to use. The question still remains why a developer of an application should make use of this API to store meaningful metadata associated to files. From the authors' perspective, the need for file metadata is obvious: the usage of desktop search engines, which usually provide mechanisms to index and retrieve structured file annotations, is very common. In the majority of cases these search engines rely on extraction plugins, which must be customized to file formats in order to extract semantically meaningful information for indexing.

Hence we can observe an encoding/decoding pattern: potentially relevant file metadata are written and encoded into file contents, and must later be decoded by metadata extractors for retrieval purposes. The sile model, its virtual file system representation, and its API close this gap by providing the infrastructure for storing, maintaining, and retrieving annotations in a format- and platform-independent manner. It abstracts over concrete metadata representation mechanisms that can be found in modern file systems (cf. Section 2), and it integrates with Semantic Web technologies. Thus, it can be considered as a step towards improved metadata interoperability in the domains of personal and social information management.

For all usage patterns outlined in Section 3 a semantically-enriched file system can bring significant benefit. User-driven file structures can be enriched (or even entirely replaced) by tagging structures, which avoid several problems of strictly hierarchical organization structures. Richer user interfaces to browse and search files can be realized using a metadata-centric approach, e.g., faceted browsing or tag clouds, while the traditional, well-known tree-based navigation metaphor can still be emulated. The same applies to application-driven and hidden file structures, which are usually anyway accompanied with application-specific metadata structures. By making hidden metadata explicit, but at the same time disassociating them from physical storage attributes, a flexible metadata-centric information space can be established. Since the different types of information can be separated orthogonally (e.g., by using sile category annotations) this space may encompass the entire information sphere of a user, which still can be efficiently managed due to its rich associated metadata.

Vocabularies for Desktop Data. Since the sile model, like RDF, uses URIs to identify annotations, it is obvious to reuse existing vocabularies and ontologies from the Semantic Web for interoperability purposes. Although, due to the open design of the sile model, an application can freely define terms, the usage of shared and commonly accepted vocabularies is strongly recommended. Shared vocabularies enable other applications—either on the same machine, or when siles are exchanged across systems—to interpret annotations in a semantically correct way. To establish that kind of interoperability, we propose the following strategy for sile annotation vocabularies:

1. Whenever possible, use terms taken from widely used vocabularies that are published on the Web in a structured, machine-readable format, i.e., RDFS or OWL.
2. If there is no such term that reflects the required semantics, reuse a semantically broader term by establishing e.g., an `rdfs:subPropertyOf` relationship, refine its semantics for the target application context within a new namespace, and publish it on the Web. This well-known procedure originating from the metadata area [19] allows one to create context-bound application profiles with clear and more specific semantic definitions.
3. If (1) and (2) are not feasible, create a suitable vocabulary and publish it on the Web in order to make it accessible also for other users and applications.

There already exist a number of widely used vocabularies, many of which are applicable for desktop data. Semantic search engines, such as Sindice [24] and Swoogle [11], or index sites for the Semantic Web[8] are good starting points to search for existing vocabularies. Figure 8 shows a representative set of Semantic Web vocabularies that are relevant for the desktop, grouped by their application domain. For each vocabulary we also indicate their base language as well as the number of concepts and properties they define.

Our analysis indicates that the Semantic Web already provides a large number of vocabularies, which cover a large share of the data we find on typical desktops. Many of the vocabularies included in this analysis are compact in terms of the number of classes and properties, and hence are relatively easy to understand and to implement. Moreover, a number of these vocabularies have been defined by re-using terms from other vocabularies. For instance, the *Description of a Project (DOAP)* vocabulary is based on *Friend-of-a-Friend (FOAF)* and therefore each application that understands FOAF is also enabled to interpret DOAP-based data to a certain extent.

The majority of the vocabularies presented in Figure 8 are actually widely used, especially in the context of Linked Data [4]. Many data providers, both from scientific and commercial domains, make use of these vocabularies to expose data about millions of resources. Similarly, the NEPOMUK ontologies are already used by a number of desktop applications that are built on top of the Nepomuk-KDE framework (cf. Section 2). Based on their concrete usage we can assume that they are suitable for their respective application domains.

6 Implementation

We have realized a sile-based file system on top of our SemDAV semantic repository[9], which implements the sile model and uses a combination of an RDF store

[8] e.g., `http://pingthesemanticweb.com/stats/namespaces.php`
[9] `http://www.semdav.org`

Vocabulary Name	Base	Concepts	Props.
General, Documents			
Dublin Core (DC)[a]	RDFS	22	55
NEPOMUK Annotation Ontology (NAO)[b]	RDFS	4	31
NEPOMUK File Ontology (NFO)[c]	RDFS	47	60
Contacts, Communication			
Friend of a Friend (FOAF)[d]	OWL	12	54
Semantically Interlinked Online Communities (SIOC)[e]	OWL	11	53
NEPOMUK Contact Ontology (NCO)[f]	RDFS	30	55
NEPOMUK Message Ontology (NMO)[g]	RDFS	7	23
VCard Ontology[h]	OWL	5	54
Calendar and Events, Project Management			
Description of a Project (DOAP)[i]	RDFS	7	30
RDF Calendar Schema[j]	OWL	14	48
NEPOMUK Calendar Ontology (CAL)[k]	RDFS	51	107
Location			
WGS84 Geo Positioning[l]	RDFS	2	4
GeoNames Ontology[m]	OWL	7	18
Multimedia			
Music Ontology[n]	OWL	53	131

[a] `http://purl.org/dc/terms/`
[b] `http://www.semanticdesktop.org/ontologies/2007/08/15/nao#`
[c] `http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#`
[d] `http://xmlns.com/foaf/0.1/`
[e] `http://rdfs.org/sioc/ns#`
[f] `http://www.semanticdesktop.org/ontologies/2007/03/22/nco#`
[g] `http://www.semanticdesktop.org/ontologies/2007/03/22/nmo#`
[h] `http://www.w3.org/2006/vcard/ns#`
[i] `http://usefulinc.com/ns/doap#`
[j] `http://www.w3.org/2002/12/cal/ical#`
[k] `http://www.semanticdesktop.org/ontologies/2007/04/02/ncal#`
[l] `http://www.w3.org/2003/01/geo/wgs84_pos#`
[m] `http://www.geonames.org/ontology#`
[n] `http://purl.org/ontology/mo/`

Fig. 8: Relevant Semantic Web vocabularies for the desktop

(Jena, SDB, PostgreSQL) and plain files to persist siles and their annotations. It exposes the stored siles via a variety of protocols and interfaces (including XML-RPC, RMI, WebDAV, HTTP, and SPARQL).

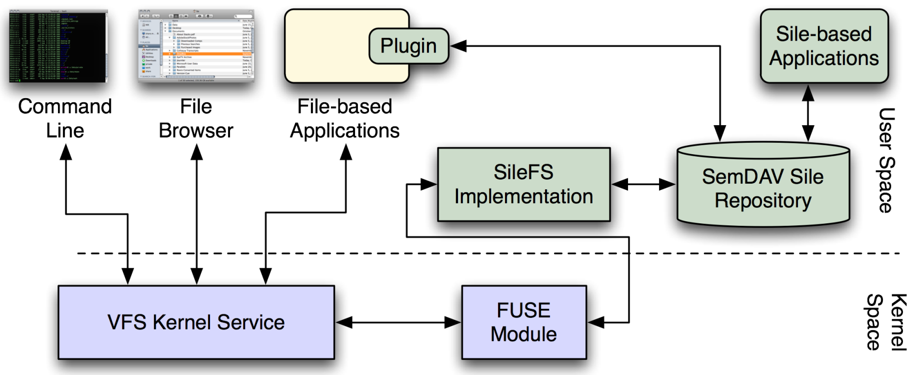

Fig. 9: Virtual File System Architecture

In addition to this repository implementation, we have developed a component that handles file system calls that are forwarded from the local machine's kernel and translates them to corresponding queries and operations on the sile model, according to the file system representation described in Section 4. To integrate the system with the local file system we have used the FUSE framework[10] and its Java binding FUSE-J[11]. FUSE defines a set of interfaces and data structures that describe files, their metadata structures, and operations thereon. Based on these frameworks we simulate a local file system that can be accessed by applications and users as if it was a common file system; at the same time the files can also be annotated and queried through the sile API. Since all files in this virtual file system are persisted as siles in our repository and hence can be accessed only through the sile API or through the virtual file system, data consistency and completeness is ensured at all times. The architecture of this implementation is depicted in Figure 9.

Since file paths are represented as explicit sile annotations, it is straightforward to implement extensions or plug-ins for existing file-based applications: whenever an application operates on a file that is stored on a sile-backed file system, the corresponding sile can be easily retrieved, and vice versa.

[10] `http://fuse.sourceforge.net`
[11] `http://fuse-j.sourceforge.net`

7 Performance Evaluation

To evaluate the performance of our approach, we have analyzed the execution times of typical file system operations. To estimate a realistic amount of data, we crawled the home directories of our department's members, which includes scientific staff (7 persons) as well as technical and administrative staff (3 persons). We used only home directories in favor of scanning entire hard disks because personal data will be the target domain for a semantic file system, and there is little need to semantically annotate system- and application-internal file structures. We discarded files that were on a black list of files and directories that usually are present in users' home directories but are not directly accessed by end users; e.g., `.svn`, `desktop.ini`, and `*.tmp`. The resulting average size of the home directory was 38,000 files stored within 5,150 directories. We consider these numbers as upper limits, since we assume that the home directories of computer scientists will typically contain more files (e.g., source code trees) than those of average end users.

Dataset #	**1**	**2**	**3**
Hierarchy depth	2	3	4
Average no. of sub-directories per directory	5	6	7
Average no. of files per directory	12	15	15
Total number of siles (directories and files)	403	4,144	44,816
Total number of RDF triples	3,626	37,295	403,343
Total number of RDF triples incl. ontologies	4,361	38,030	404,078

Fig. 10: Datasets for performance evaluation

To estimate the influence of the size of home directories on our system's performance, we artificially created three test data sets, which are described in Figure 10. To represent basic data about files and directories (cf. Section 3) nine triples per object were created. Note that this does not include any additional descriptive triples (i.e., semantic annotations); these were not considered in our performance evaluation. Our implementation also requires loading a set of core ontologies, which add another $\approx$700 triples to the database.

Virtual File System-Based Access. We have analyzed the runtime performance of typical file system access patterns: navigation between directories, listing of directory contents, deletion, moving, and renaming of files. We have carried out the experiments on a high-end consumer notebook (MacBook Pro, Core 2 Duo, with 2 GB RAM) running Mac OS X 10.5 and JVM 1.5. We have used the command shell (`/bin/bash`) to perform our measurements and used only standard commands (`cd`, `ls`, `rm`, and `mv`). Because of our implementation architecture, each operation is processed by a number of components that are not under our direct control (e.g., the FUSE kernel module; see also Figure 9). Hence we do not have influence on how shell commands are translated to file system driver calls; for instance, issuing a directory listing command (`ls`) causes the execution of four distinct FUSE calls being passed to our implementation. Nevertheless, our goal was to measure the execution time as experienced by the end user, hence we tracked the total processing time of commands, including overhead caused by the operating system and the FUSE kernel module.

The operations we have evaluated involve read-only access (directory navigation and directory listing) and read+write operations (deletion, moving, renaming). For the latter, the complexity of read and write operations differs: for a sile deletion, (*1*) the triples within the store that describe the object to be deleted have to be identified (read), and (*2*) these triples have to be removed from the store (write). Move and rename operations require in principle the same access operations, whereas a move across directories requires an additional read and write operation, namely the update of the relationship between the file and its parent directory. We did not evaluate the performance of operations that affect multiple siles (e.g., moving an entire file system subtree) since we are aware of the fact that the current modeling approach does not support such operations in a satisfying way. We plan to work on a more efficient support for such operations in the future. Further, we did not evaluate the performance of actual read and write operations on the file content: the modifications to metadata caused by these actions are comparable to those of a move operation (i.e., an update of the `content-length` and `update-time` properties), and the actual file content is provided by the underlying (physical) file system and hence is out of the scope of our performance measurements.

Dataset #	**1**	**2**	**3**
Total number of siles	403	4,144	44,816
`cd`	0.029	0.048	0.107
`rm`	0.063	0.142	0.879
`ls`	0.258	0.464	1.547
`mv` within directory	0.254	0.488	2.488
`mv` across directories	0.296	0.688	3.238

Fig. 11: Virtual file system access operations: average execution times in seconds

For our experiments, we executed every operation 10 times in random order, and repeated the entire experiment five times. The results of our experiments are depicted in Figure 11. For the first two datasets (≈400 and ≈4,000 siles) we can observe very low execution times, which allow for uninterrupted interactive work with virtual file systems. For a dataset consisting of ≈40,000 siles, the response times for simple operations (change directory, remove file) are still in a reasonable range, and even the operations that involve multiple, complex queries (directory listing, moving) can be performed in a time that is comparable to web browsing and acceptable for interactive use. These results indicate that even a prototypical implementation of a virtual file system, based on our data model and built using an off-the-shelf RDF triple store without further optimization, provides acceptable performance for everyday usage on a typical consumer machine. A semantic file system based on a more efficient triple store that is better integrated into the operating system could achieve even better performance, since this would allow us to circumvent the rather inefficient architecture that we have chosen for the sake of implementation simplicity.

Metadata-Based Access. In addition to methods that are executed through the virtual file system interface, we also analyzed the execution runtime of typical queries that retrieve siles based on arbitrary combination of metadata. These queries cannot be directly executed through the virtual file system; instead, the sile API (cf. Section 5) and a corresponding hierarchy of filters was used for these test runs. We extended the datasets from the previous section with sile annotations, whereas each sile was annotated with 20 annotations (tags, attributes, categories, and slinks) in average. These additional metadata amplified our test data sets up to 1.2 million triples for 40,000 siles. The requests were issued via our prototype's Java interface (cf. Section 6); each group of queries was repeated 10 times in random order.

Dataset #	**1**	**2**	**3**
Total number of triples	11,049	123,967	1,238,534
Search siles with a specific tag	0.271	1.084	2.008
Search siles with one out of three tags	0.734	1.182	2.304
Search siles related to a given sile	0.014	0.029	0.039
Retrieve all annotations for one sile	0.037	0.050	0.071
Create one sile	0.131	0.134	0.175
Add one tag to a sile	0.122	0.134	0.153
Delete one sile	0.044	0.051	0.071

Fig. 12: Metadata-based access operations: average execution times in seconds

The evaluated operations include search requests of varying complexity, ranging from searches for siles that are tagged with a single tag, to OR-combined search criteria. Additionally we tested calls that retrieved all annotations for a given sile, and operations that create and delete siles, respectively. The results (cf. Figure 12) accentuate the performance overhead caused by the relatively complex architecture of our virtual file system, as described before. Operations that are executed directly against the sile repository are executed significantly faster than operations that are issued through the virtual file system. This strengthens the need for a more efficient implementation of a virtual file system driver that is directly coupled to the sile repository, and thus avoids the indirection caused by our current prototypical architecture.

8 Conclusions

We discussed a number of issues regarding the integration of semantic technologies with file systems, which is a crucial requirement for a successful deployment of Semantic Desktop solutions. First, we showed that the RDF data model exposes a number of characteristics that may cause problems when used in the context of information management on the desktop. To overcome these limitations, we proposed the sile model, which combines characteristics from both the Semantic Web and file systems. Siles are digital objects that have a globally valid, immutable identity and can be annotated by tags, categories, and attributes; furthermore, they can be semantically related to each other. This model provides an integrated view on desktop resources and associated semantic annotations and is intended to serve as an intermediate layer between applications and the actual storage infrastructure. In conjunction with this model we presented an Application Programming Interface that allows developers to manipulate and retrieve siles and their annotations.

We also discussed our strategy for representing files and directories using siles and sile annotations. This allows us to simulate the behavior and characteristics of traditional, hierarchical file systems, which are used by a magnitude of applications. By providing a virtual file system view on a sile repository, users have the continuing ability to use the applications they are familiar with. In contrast to other semantic file systems, which expose semantic annotations as virtual directories, we employ designated attributes for this purpose. Therefore, we can simulate the behavior of file systems more accurately than other approaches since (virtual) file paths are not depending on semantic annotations, which may change over time and thus make path-based file references invalid.

The sile model is designed to be compatible with Semantic Web technologies. Therefore also analyzed—as a further contribution—a representative set of vocabularies that can be used to annotate siles without further modification. These vocabularies cover a large fraction of the semantic definitions needed for desktop data. We proposed strategies how such vocabularies can be used in the desktop context in order to foster data interoperability on a global scale.

Finally, we presented our RDF-based implementation of the sile model and a virtual file system that is backed by our system. We analyzed the performance of typical file system operations under the consideration of realistic amounts of data that can be found on typical users' desktops, and demonstrated that the performance of such a virtual file system is acceptable for interactive usage. We also showed that the performance of metadata-based access to sile repositories is acceptable for typical desktop usage, even under the restrictions of a prototypical implementation.

Acknowledgements Parts of this work have been funded by FIT-IT grants 812513 and 815133 from the Austrian Federal Ministry of Transport, Innovation, and Technology. The authors thank Stefan Pomajbik, Diman Todorov, and Arash Amiri for their support in the implementation of the system, and the anonymous reviewers for their helpful and inspiring comments.

References

1. Sasha Ames, Nikhil Bobb, Kevin M. Greenan, Owen S. Hofmann, Mark W. Storer, Carlos Maltzahn, Ethan L. Miller, and Scott A. Brandt. LiFS: An Attribute-Rich File System for Storage Class Memories. In *Proceedings of the 23rd IEEE / 14th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2006.
2. Mark Birbeck and Shane McCarron. *CURIE Syntax 1.0 — A Syntax for Expressing Compact URIs*. World Wide Web Consortium, http://www.w3.org/TR/curie/, 2009. Available at `http://www.w3.org/TR/curie`.
3. Chris Bizer, Richard Cyganiak, and Tom Heath. *How to Publish Linked Data on the Web*, 2007. Available at `http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/`, retrieved 02-Dec-2008.
4. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data — The Story So Far. *International Journal on Semantic Web and Information Systems*, to appear.
5. Stephan Bloehdorn, Olaf Görlitz, Simon Schenk, and Max Völkel. TagFS – Tag Semantics for Hierarchical File Systems. In *6th International Conference on Knowledge Management (I-KNOW'06)*, 2006.
6. Lukas Blunschi, Jens-Peter Dittrich, Olivier René Girard, Shant Kirakos Karakashian, and Marcos Antonio Vaz Salles. A Dataspace Odyssey: The iMeMex Personal Dataspace Management System. In *Third Biennial Conference on Innovative Data Systems Research*, pages 114–119, 2007.
7. Richard Boardman. Multiple Hierarchies in User Workspace. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, pages 403–404, New York, NY, USA, 2001. ACM Press.
8. Yuhan Cai, Xin Luna Dong, Alon Halevy, Jing Michelle Liu, and Jayant Madhavan. Personal Information Management with SEMEX. In *Proceedings of the 2005 ACM SIGMOD Conference on Management of Data*, pages 921–923, New York, NY, USA, 2005. ACM Press.
9. Michael J. Carey, David J. DeWitt, Joel E. Richardson, and Eugene J. Shekita. Object and File Management in the EXODUS Extensible Database System. In *VLDB '86: Proceedings of the 12th International Conference on Very Large Data Bases*, pages 91–100, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.
10. Stefan Decker and Martin R. Frank. The Networked Semantic Desktop. In *WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web*, 2004.
11. Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: A Search and Metadata

Engine for the Semantic Web. In *CIKM '04: Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 652–659, New York, NY, USA, 2004. ACM Press.

12. Paul Dourish, W. Keith Edwards, Anthony LaMarca, and Michael Salisbury. PRESTO: An Experimental Architecture for Fluid Interactive Document Spaces. *ACM Trans. Comput.-Hum. Interact.*, 6(2):133–161, 1999.
13. Thomas Franz, Ansgar Scherp, and Steffen Staab. Are Semantic Desktops Better? — Summative Evaluation Comparing a Semantic against a Conventional Desktop. In *Fifth International Conference on Knowledge Capture (K-CAP 2009)*, 2009.
14. Thomas Franz, Steffen Staab, and Richard Arndt. The X-COSIM Integration Framework for a Seamless Semantic Desktop. In *K-CAP '07: Proceedings of the 4th International Conference on Knowledge Capture*, pages 143–150, New York, NY, USA, 2007. ACM.
15. David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole Jr. Semantic File Systems. In *SOSP '91: Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 16–25, New York, NY, USA, 1991. ACM Press.
16. Richard Grimes. Code Name WinFS: Revolutionary File Storage System Lets Users Search and Manage Files Based on Content. *MSDN Magazine*, 19(1), 2004.
17. Tudor Groza, Siegfried Handschuh, Knud Moeller, Gunnar Grimnes, Leo Sauermann, Enrico Minack, Cedric Mesnage, Mehdi Jazayeri, Gerald Reif, and Rosa Gudjonsdottir. The NEPOMUK Project — On the Way to the Social Semantic Desktop. In Tassilo Pellegrini and Sebastian Schaffert, editors, *Proceedings of I-Semantics' 07*, pages pp. 201–211. JUCS, 2007.
18. Patrick Hayes. *RDF Semantics (W3C Recommendation 10 February 2004)*. World Wide Web Consortium, 2004.
19. Rachel Heery and Manjula Patel. Application Profiles: Mixing and Matching Metadata Schemas. *Ariadne*, (25), 2000.
20. David R. Karger. Haystack: Per-User Information Environments Based on Semistructured Data. In Victor Kaptelinin and Mary Czerwinski, editors, *Beyond the Desktop Metaphor*, pages 49–100. Massachusetts Institute of Technology, 2007.
21. Ben Martin. The World is a libferris Filesystem. *Linux Journal*, April 2006.
22. Sergio Muñoz, Jorge Pérez, and Claudio Gutiérrez. Minimal Deductive Systems for RDF. In *The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings*, 2007.
23. Adaora Okoli and Bernhard Schandl. Extraction of Contextual Metadata from File System Interactions. In *Workshop on Exploitation of Usage and Attention Metadata (EUAM 09) Workshop on Exploitation of Usage and Attention Metadata (EUAM 2009)*, 2009.
24. Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com — A Document-oriented Lookup Index for Open Linked Data. *Int. J. Metadata Semant. Ontologies*, 3(1):37–52, 2008.
25. Yoann Padioleau, Benjamin Sigonneau, and Olivier Ridoux. Lisfs: A logical information system as a file system. In *Proceeding of the 28th International Conference on Software Engineering (ICSE 2006)*, pages 803–806, New York, NY, USA, 2006. ACM Press.
26. Jörg Richter and Jurij Poelchau. DeepaMehta — Another Computer is Possible. In Jörg Rech, Björn Decker, and Eric Ras, editors, *Emerging Technologies for*

Semantic Work Environments: Techniques, Methods, and Applications. Idea Group Inc., 2008.

27. Leo Sauermann, Ansgar Bernardi, and Andreas Dengel. Overview and Outlook on the Semantic Desktop. In Stefan Decker, Jack Park, Dennis Quan, and Leo Sauermann, editors, *Proceedings of the 1st Semantic Desktop Workshop*, volume 175, Galway, Ireland, November 2005. CEUR Workshop Proceedings.
28. Leo Sauermann and Dominik Heim. Evaluating Long-Term Use of the Gnowsis Semantic Desktop for PIM. In *The Semantic Web — ISWC 2008*, volume 5318 of *LNCS*, pages 467–482. Springer, 2008.
29. Bernhard Schandl. *An Infrastructure for the Development of Semantic Desktop Applications*. PhD thesis, University of Vienna, Department of Distributed and Multimedia Systems, 2009.
30. Bernhard Schandl. Representing Linked Data as Virtual File Systems. In *Proceedings of the 2nd International Workshop on Linked Data on the Web (LDOW), Madrid, Spain*, 2009.
31. Bernhard Schandl and Bernhard Haslhofer. The Sile Model – A Semantic File System Infrastructure for the Desktop. In *Proceedings of the 6th European Semantic Web Conference (ESWC 2009), Heraklion, Greece*, 2009.
32. Bernhard Schandl and Ross King. The SemDAV Project: Metadata Management for Unstructured Content. In *CAMA '06: Proceedings of the 1st International Workshop on Contextualized Attention Metadata: Collecting, Managing and Exploiting of Rich Usage Information*, pages 27–32, New York, NY, USA, 2006. ACM Press.
33. Bernhard Schandl and Niko Popitsch. Lifting File Systems into the Linked Data Cloud with TripFS. In *3rd International Workshop on Linked Data on the Web (LDOW2010) - Raleigh, North Carolina, USA*, 2010.
34. Florian Schmedding, Christoph Hanke, and Thomas Hornung. RDF Authoring in Wikis. In Christoph Lange 0002, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the 3rd Semantic Wiki Workshop (SemWiki 2008)*, volume 360 of *CEUR Workshop Proceedings*, 2008.
35. Stuart Sechrest and Michael McClennen. Blending Hierarchical and Attribute-Based File Naming. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 572–580. IEEE Computer Society, 1992.
36. Michael Sintek, Ludger van Elst, Simon Scerri, and Siegfried Handschuh. Distributed Knowledge Representation on the Social Semantic Desktop: Named Graphs, Views and Roles in NRL. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *Proceedings of the 4th European Semantic Web Conference (ESWC 2007), Innsbruck, Austria*, volume 4519 of *Lecture Notes in Computer Science*, pages 594–608. Springer, 2007.
37. Michael Sintek, Ludger van Elst, Simon Scerri, and Siegfried Handschuh. NEPOMUK Representational Language Specification. Technical report, NEPOMUK Project Consortium, 2007. Available at `http://www.semanticdesktop.org/ontologies/nrl`, retrieved 02-Dec-2008.
38. C.E. Wills, D. Giampaolo, and M.S. Mackovitch. Experience with an Interactive Attribute-based User Information Environment. In *Computers and Communications, 1995. Conference Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on*, pages 359–365, Mar 1995.