The Sile Model — A Semantic File System Infrastructure for the Desktop

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, and 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 the sile model, an intermediate data model that combines attributes of the Semantic Web and file systems. This model is intended to be the conceptual foundation of the Semantic Desktop, and to serve as underlying infrastructure on which applications and further services, e.g., virtual file systems, can be built. In this paper, we present the sile model, discuss Semantic Web vocabularies that can be used in the context of this model to annotate desktop data, and analyze the performance of typical operations on a virtual file system implementation that is based on this model.

1 Introduction

A large amount of information is 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 increasing constantly, 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 silos, and—contrary to the Web, where hyperlinks between documents, and across site boundaries, can be defined—there exist only limited means to define and retrieve relationships between different resources on the desktop.

The Semantic Web tries to deal with the problems mentioned before by adding a layer on top of the existing Web infrastructure, in which descriptions

about web resources are expressed in 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 on the Semantic Web, forming a solid data corpus that can be crawled by (semantic) search engines, or serve as foundation for applications.

Recent research in the field of the Semantic Desktop [2, 10, 12] has shown that a number of characteristics of 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, and the flexible usage of formalized vocabularies to describe resources. It has been shown [15] 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 differences between the Web and the desktop contexts. First, in contrast to the World Wide Web, the desktop already has a *de facto* organization metaphor for data: hierarchical file systems. File hierarchies have been in use for decades, and the vast majority of personal information is stored in these structures. 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. As a second difference, while it is accepted on the Web that URLs may be broken and web resources appear and disappear, this is not the case for the desktop. Users rightfully expect their data to remain consistent and complete at all times.

Since the RDF data model has by design a number of shortcomings that may cause problems when a Semantic Desktop is to be efficiently implemented (see Section 2), we propose in this paper the *sile model*, a data model that acts as an intermediate layer between file systems and Semantic Web technology. This model allows users and applications to annotate and interrelate desktop resources, but preserves the possibility to be used as a hierarchical file system. Thus, it maintains full backwards-compatibility to existing systems and applications.

Despite the number of existing differences between the sile model and RDF, the former has been designed to be interoperable with vocabularies from the Semantic Web. Interoperability between the desktop and the Web is desirable in order to provide a unified data space to the user. Tools and applications are enabled 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 can refer to Semantic Web resources, as identifiers for all annotations in our data model.

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

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

2 Integrating File Systems and RDF Descriptions

File Systems as 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 the users to organize files in directories named and nested according to their needs. Examples for this group include applications like word processors or spreadsheet tools. A second group of applications establish *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 like Mozilla Thunderbird or media players like iTunes. Typically, these applications do not operate on single files but on file collections, and interpret the directory hierarchies and file names in a schematic manner. A third group of applications do not expose a file structure 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.

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 [3]; and second, it extends the possibilities provided by file systems with respect to search and retrieval [15]. All three classes of applications, as described before, could benefit from such an integration: user-driven file structures could be extended by more powerful and non-hierarchical annotation mechanisms, like tagging 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, 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 [1] indicates that it is sufficiently expressive for a large

number of applications, ranging from statistical data to geo information and multimedia content. Certain RDF characteristics, however, are problematic in the context of the desktop: for instance, 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 (cf. [11], Section 1.2). In the open Web environment, it is acceptable for users and applications that resources may become unavailable and links may become broken. However, on the desktop, integrity of content objects and their annotations 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, and hence recent approaches to Semantic Desktop information modelling have restricted the interpretation of RDF data to a closed world semantics (e.g., [18]).

As a third problematic issue, the application of certain RDF language elements (in particular collections, blank nodes, and reification), has been discouraged both in the context of the Web (cf. [1], Section 2.2) and the desktop (cf. [19], Section 2.3.2). These features significantly increase the complexity of RDF processing, as well as representing RDF in user interfaces [13, 17], but expose a very light semantics [13]. 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., [2, 10, 12]) 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. For instance, a URI that refers to a file in the user's home directory could be `file:////home/bernhard/work/papers/eswc2009-submission.pdf`. This URI, however, becomes invalid when a file is renamed, moved to a different directory, 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, where such propagation is not possible, the applicability of the entire system is heavily limited.

In the following, we present an alternative approach that aims to solve the problems described before: we consider an object's content and its annotations as integral components, which are always processed together. Instead of adding a semantic layer on top of existing structures, we inject the Semantic Web into the core of the data representation structures—in our case, the file system—, and provide a virtual, file system-like representation of the data stored therein which can be accessed by existing desktop applications.

3 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 used to retrieve the resource's representation (cf. Figure 1(a)), this is not 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 1(b).

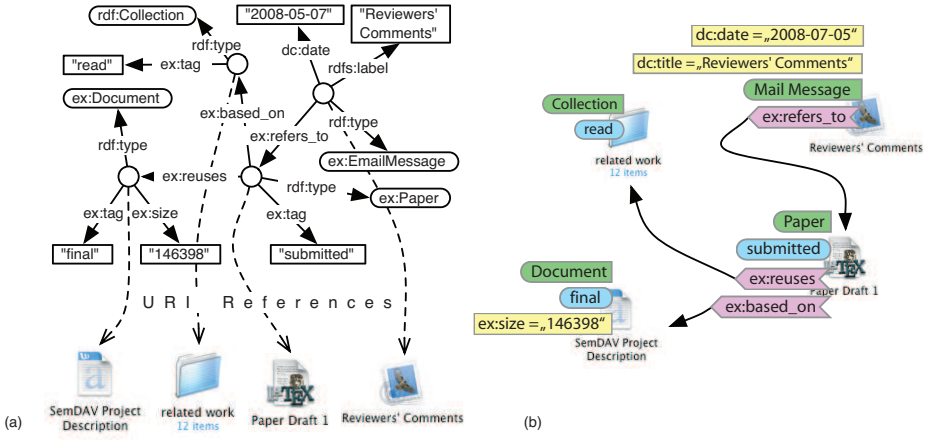

Fig. 1: (a) RDF model: URIs refer to actual content; (b) Sile model: integrated view on content and annotations.

In our model we distinguish between four types of annotations, which from our experience 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 ontology); *attributes* in the form of typed name/value pairs; and *slinks* (semantic links), i.e., directed typed relationships between siles. Categories, attributes, and slinks are identified by URIs, which allow for a non-ambiguous interpretation of their semantics; however the formalism used for this purpose is out of the scope of this definition. A category annotation, for instance, may refer to an OWL ontology class as well as to a table within a relational database schema. Figure 1 depicts a number of siles and their associated annotations, whereas different shapes and colors are used to indicate different annotation types.

Because our model relies on URIs to identify siles and annotations, it is straightforward to represent sile annotations in RDF. Figure 2 depicts an ex-

```
<urn:uuid:57207370-6880-11dd-ad8b-0800200c9a66>
  a sile:Sile ;
  sile:label "SemDAV Project Description"^^xsd:string ;
  sile:cat-type nfo:PaginatedTextDocument ;
  sile:tag-type sile-tag:final ;
  sile:creation-date "2008-07-11T16:21:14"^^xsd:dateTime ;
  sile:update-date "2008-07-11T17:14:21"^^xsd:dateTime ,
                   "2008-07-11T17:32:02"^^xsd:dateTime ;
  sile:content-type "application/msword"^^xsd:string ;
  sile:content-length "146398"^^xsd:long ;
  dc:subject <http://www.semdav.org> ;
  sile:cat-type silefs:File ;
  silefs:path "/home/projects/semdav/docs/semdav_description.doc"
        ^^xsd:string ;
  silefs:parent <urn:uuid:60ad6a73-1b60-4553-9436-d09d395fc29c> .
```

Fig. 2: RDF representation of a sile (N3 notation)

ample of a sile that represents a project report. This sile is annotated with a number of attributes (like label, creation date, content length, and subject), one category (`nfo:PaginatedTextDocument`), and one tag (`final`).

Representation of Directory Hierarchies. We can now use our data model to represent all elements of a file system without loss of information. A basic file system can be represented by an unordered tree having two types of nodes: directories and files. Directory nodes may have children, whereas file nodes are always leaf nodes. Each node is labelled with a local name (i.e., the file or directory name), which is unique within the scope of its parent.

We map each node in the directory tree to a sile and represent its type by a category. We further represent the node name as an attribute annotation. 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. This requires processing overhead when modifications to the file tree are made, but significantly increases the performance of read operations.

The RDF representation of these additional sile annotations is shown in the last three lines of Figure 2. These annotations identify the resource as a file, describe its file name and absolute path, and relate the resource to its parent directory. This representation allows for efficient execution of typical file system operations. For most such operations the full paths of involved files are given, which allows us to retrieve the corresponding sile with a simple query for the `silefs:path` attribute. Operations that involve a directory and its children (e.g., a directory listing) can be resolved by querying for `silefs:parent` relationships

```
Sile docSile = repo.getSileForFile("/home/projects/semdav/docs/
   semdav_description.doc");
docSile.addAnnotation(repo.getCategory("nfo:PaginatedTextDocument"));
docSile.addAnnotation(repo.getTag("final"));
Sile paperSile = repo.getSileForFile("/home/papers/eswc2009/paper.tex");
paperSile.addAnnotation(repo.getSlink("ex:based_on", docSile));
```

Fig. 3: Sile API Code Example

between siles. When a file or directory is renamed or moved[3], 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 also be updated to point 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 enables us also to represent hard 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.

4 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 used by any application without modifications. To read, write, and search for semantic annotations of files that are represented as siles we have developed an Application Programming Interface (API). A detailed discussion of this API is outside the scope of this paper; however to give the reader an impression of the structure and usage of the API, a code snippet is depicted in Figure 3. This example reproduces the annotations depicted in Figure 1 and Figure 2: a category and a tag are attached to a file (`"semdav_description.doc"`). Later, the application (or the user) could easily retrieve the object by searching for these annotations.

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 annotations. Additionally, the API allows an application to search for siles that match certain criteria and to retrieve the corresponding file paths. For instance, a word processing application could be extended so that it stores metadata about created documents (e.g.,

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

author and title) as sile attributes, or a search operation could be used to retrieve files that are associated with a certain project.

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 2, 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, either for single applications or operating systems.

Vocabularies for Desktop Data. Since the sile model is closely related to the RDF model and 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 preferred. 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 it for the purpose of the target application within a new namespace, and publish it on the Web.
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 [14] and Swoogle [5], or index sites for the Semantic Web[4] are good starting points to search for existing vocabularies. In Figure 4 we present 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 we have included in our 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,

[4] e.g., `http://pingthesemanticweb.com/stats/namespaces.php`

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. 4: Relevant Semantic Web Vocabularies for the Desktop

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.

5 Implementation Case Study

We have realized a virtual file system on top of our SemDAV semantic repository[5], which implements the sile model and uses a combination of an RDF store (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, and HTTP).

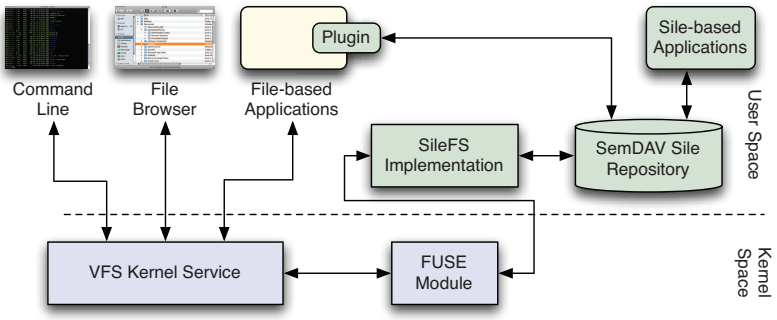

Fig. 5: Virtual File System Architecture

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 3. To integrate the system with the local file system we have used the FUSE framework[6] and its Java binding FUSE-J[7]. 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 which can be accessed by applications and users as if it were a common file system; but 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 5.

[5] `http://www.semdav.org`
[6] `http://fuse.sourceforge.net`
[7] `http://fuse-j.sourceforge.net`

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.

6 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 view 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. 6: 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 6. To represent basic data about files and directories (cf. Section 2) 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 ≈700 triples to the database.

We have analyzed the runtime performance of typical access patterns to file systems: 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 external components (e.g., the FUSE kernel module; see also Figure 5) which are not under our direct control. 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 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. For our experiment, we have executed each of these operations 10 times in random order, and the entire experiment was repeated five times.

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. 7: Evaluation Results: Average Execution Time in Seconds

The results of our experiments are depicted in Figure 7. 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 operations that involve multiple, complex queries (directory listing, moving) are within a range comparable to accessing remote file systems via the Web. 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 file system and hence is out of the scope of our performance measurements.

These numbers 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, has acceptable performance for everyday usage on a typical consumer machine. A semantic file system and a more efficient triple store, more tightly 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.

7 Related Work

The idea of adding semantic technologies to file systems has been widely researched. The benefits of virtual file systems based on semantically enriched data have been discussed and demonstrated in a number of works [4, 6–9, 16]. Usually, semantic annotations of files (e.g., tags, ontology classes, or property/value pairs) are mapped to virtual directories which contain the resources that match the criteria. This often implies however that a resource may be accessible through multiple paths, and also may induce problems when files are written to virtual directories, as the mapping between directory names and semantic annotations is often not bijective (e.g., when disjunctions are included in the underlying query). In the case of large datasets and extensive annotations, virtual hierarchies may become very large and therefore difficult to understand and browse by the user. Our approach does not attempt to perform such a mapping; instead we use explicit annotations for directory structures. Hence we preserve full compatibility with existing applications and tools that are implemented under basic assumptions about the structure of file systems (e.g., each resource exists at exactly one place), which is a crucial requirement for applications built on top of file systems; e.g., desktop search engines.

A number of Semantic Desktop projects [2, 10, 12] aim to provide a semantic infrastructure that covers all applications and information needs that users operate on. These approaches often use file system crawlers to incorporate file systems into the personal information space. As we have described in Section 2, this may cause problems; therefore we consider our work a complementary virtual file system that may be integrated into a Semantic Desktop.

8 Conclusions

We have 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 have proposed the sile model, which combines characteristics from both the Semantic Web and file systems. This model provides an integrated view on desktop resources and associated semantic annotations, and it is intended to serve as an intermediate layer between applications and actual storage infrastructure. Second, we have analyzed a representative set of Semantic Web vocabularies and showed that the Semantic Web already provides a sufficient number of suitable vocabularies to be used in the desktop context. Third, we presented our RDF-based implementation of the sile model and a virtual file system that is backed by our system. We have 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 aim to more tightly integrate our approach with common desktop operating systems, from which we expect significant performance improvements.

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 support in the implementation of our system, and Stefan Zander and Wolfgang Jochum for their valuable comments on this paper.

References

1. 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.
2. 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 *CIDR*, pages 114–119. www.crdrdb.org, 2007.
3. Richard Boardman. Multiple Hierarchies in User Workspace. In *CHI '01: CHI '01 Extended Abstracts on Human Factors in Computing Systems*, pages 403–404, New York, NY, USA, 2001. ACM Press.
4. C. Mic Bowman, Chanda Dharap, Mrinal Baruah, Bill Camargo, and Sunil Potti. A File System for Information Management. In *Proceedings of the ISMM International Conference on Intelligent Information Management Systems*, March 1994.
5. 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 thirteenth ACM international conference on Information and knowledge management*, pages 652–659, New York, NY, USA, 2004. ACM.
6. Paul Dourish, W. Keith Edwards, Anthony LaMarca, and Michael Salisbury. Using Properties for Uniform Interaction in the Presto Document System. In *UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 55–64, New York, NY, USA, 1999. ACM.

7. Sebastian Faubel and Christian Kuschel. Towards Semantic File System Interfaces. In Christian Bizer and Anupam Joshi, editors, *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC 2008)*, volume 401. CEUR Workshop Proceedings, 2008.
8. 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.
9. Burra Gopal and Udi Manber. Integrating Content-based Access Mechanisms with Hierarchical File Systems. In *OSDI '99: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pages 265–278, Berkeley, CA, USA, 1999. USENIX Association.
10. 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.
11. Patrick Hayes. *RDF Semantics (W3C Recommendation 10 February 2004)*. World Wide Web Consortium, 2004.
12. 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.
13. 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.
14. 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.
15. 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.
16. Simon Schenk, Olaf Görlitz, and Steffen Staab. TagFS: Bringing Semantic Metadata to the Filesystem. In *Poster at the 3rd European Semantic Web Conference (ESWC)*, 2006.
17. Florian Schmedding, Christoph Hanke, and Thomas Hornung. RDF Authoring in Wikis. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the 3rd Semantic Wiki Workshop (SemWiki 2008)*, volume 360 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
18. 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.
19. 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.