Representing Linked Data as Virtual File Systems

Bernhard Schandl
University of Vienna
Department of Distributed and Multimedia Systems
Liebiggasse 4/3-4
A-1010 Wien, Austria
bernhard.schandl@univie.ac.at

ABSTRACT

One of the main characteristics of Linked Open Data (LOD) is the exclusive application of standards published and maintained by the World Wide Web Consortium. This strict adherence is kept on all levels, ranging from the identification and transportation (URI, HTTP) to the interpretation (RDF, RDFS, OWL) of resource descriptions. Because these standards are open and accessible to everybody, broad acceptance and proliferation of LOD technologies in Web-based applications and services are enabled. On typical desktops, however, the majority of applications are not aware of Web standards, but use hierarchical file systems to organize and store information. This results in a gap between the two distinct information spaces of the Web and the desktop. To bridge this gap, we propose a virtual file system representation of LOD sets, through which they can be accessed as if they were present in the file system and thus easily be used within desktop applications.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management; H.3.5 [**Information Storage and Retrieval**]: Online Information Services

General Terms

Algorithms, Design

Keywords

Linked Open Data, file systems, information representation, Semantic Desktop

1. INTRODUCTION

The goal of Linked Open Data (LOD) [5] is to increase the value of publicly available data sets by exposing them on the Web using standardized technologies, and by interlinking related items so that clients can easily combine information from various sources. To accomplish this, the LOD principles [8] are fully integrated with the Web architecture [16] and technologies: URIs are used to identify resources, RDF (usually serialized using XML) is used to describe them, and resource descriptions and representations are transferred using HTTP.

LODW April 20, 2009, Madrid, Spain.

A large number of applications, however, are executed in desktop environments and are, in turn, designed and built under entirely different assumptions w.r.t. information representation, storage, and exchange. On the desktop, the file system is the main mechanism for the storage and organization of data, and consequently data exchange on the desktop is mostly implemented based on files. This is reflected by the fact that many applications provide import and export filters, which allow them to read and write different file formats and hence exchange data with other applications. A number of de-facto standard file formats exist which are expected to work across platforms and applications.

The Semantic Web, and Linked Open Data in particular, are distinct from desktop environments in this respect. On the Web, different formats and mechanisms are in operation. As a consequence, we can observe distinct information spaces as well as conceptual and technical gaps between these two worlds. To bridge them, it is desirable to build a bridge between LOD and desktop applications so that desktop users can directly access and integrate information from LOD sources, but continue to work with the applications they are familiar with. For instance, it would be desirable to directly insert a textual description of Berlin within one's favourite word processing application, or to seamlessly load structured descriptions about this city into a spreadsheet tool.

In this paper we present such a bridge: we propose a mechanism that represents LOD sets as virtual file systems, which enables applications and users to directly access resource descriptions and their representations. Such a representation can be useful in a number of scenarios, which we describe in Section 2. In Section 3 we discuss our mapping approach and a prototypical implementation. We also discuss structural differences between file systems and LOD principles, and how the LOD technology family could be improved in order to extend their applicability in Section 4.

2. APPLICATIONS FOR VIRTUAL LOD-BASED FILE SYSTEMS

The possible usage scenarios of file systems are manifold, as we can observe on our own desktop computers. A number of them are especially interesting in the context of Linked Open Data. In this section we outline such scenarios that would benefit from a virtual representation of LOD sets.

Browsing and Navigation.

Most desktop computer users are familiar with navigation in hierarchical file systems. The visual rendering of file

system structures, provided by applications like Windows Explorer or Apple Finder, is similar on all desktop operating systems. They indicate files as atomic information entities, which are grouped by hierarchical collections, i.e., directories. Navigation within the directory hierarchy of a file system is understood by most end users: directories can be "opened" and their contents can be inspected. Similarly, files can be opened with their respective applications in order to view and manipulate their content.

A virtual file system representation of Linked Open Data applies the metaphors of directories and files to these data: it allows users to navigate through the RDF graph provided by a LOD set as if it was a hierarchical file structure on one's personal desktop. Hence users are not required to mentally "switch" between the Web and the desktop contexts.

Data Import.

The data found in LOD sources can be relevant in many scenarios. However, common desktop applications usually do not provide means to import data directly from the Web, neither in "traditional" formats (e.g., HTML pages) nor in the form of RDF. Consequently, a user of such applications who wishes to reuse information from LOD sources is forced to perform intermediate data conversion. First, the data of interest must be located, then it must be downloaded to a local file, and as a final step it must be converted into a format that can be read by the target application.

To efficiently perform these tasks, extensive knowledge of LOD technologies (SPARQL and RDF) and of the target application's data formats is needed. With a virtual file representation of LOD, at least the first two steps can be executed by the virtual file system driver, allowing users and applications to access data as if they were stored in the local file system. Additionally, conversion from RDF to typical desktop application file formats that can be interpreted by many applications (e.g., Rich Text Format or Comma-separated Values) can be performed directly by the virtual file system driver.

Integration with Desktop Resources.

Information resources on the desktop are typically organized using hierarchical file systems, which allow users to arrange documents within a tree of (nearly) arbitrarily named directories. Even applications that do not follow this pattern store and organize their data in file system structures [22]. These hierarchies help users to retrieve previously stored information, mostly by step-by-step navigation through the directory hierarchy, as described before.

The integration of resources other than files, like web resources, into file systems is often cumbersome. Many systems allow users to link web resources (URLs) into hierarchical file systems by saving the target address into a special file. This file however can often not be used by applications that operate on the file system. By representing Linked Open Data (i.e., resource descriptions on the Semantic Web) as virtual file systems, these data can be directly integrated with other file-based resources, and it can be seamlessly processed by humans and applications.

3. REPRESENTING LOD AS VIRTUAL FILE SYSTEM

3.1 Design Considerations

A number of conceptual differences between Linked Open Data and hierarchical file systems have to be considered in order to define a useful and valid representation. In the following we outline these issues and, where possible, describe directions how to solve them.

- *Structural Mismatch.* Linked Open Data is published in the RDF format, which essentially is a graph model: resources and literals can be interpreted as (labelled) nodes, and property relationships between them can be interpreted as directed, labelled edges. In contrast, file systems are trees, which consist of inner nodes (directories) and leaf nodes (files). In hierarchical file systems, each node is labelled, and there exists only one type of relationship between nodes (`contains`).

  Consequently, in order to prevent information loss, the labelled edges of the RDF graph must be represented as labelled nodes in the file system representation. A graph cannot be reduced to a tree without the loss of edges, which in the case of RDF means information loss. However, one can circumvent the strictly hierarchical structure of file systems using *shortcuts*[1]. The representation of edges in the RDF graph model as file system shortcut allows for a complete graph representation.

- *Entry Point.* File systems, due to their hierarchical structure, have a natural entry point, the *root directory*. This entry point is present in every file system and commonly serves as the starting point for activities like browsing and searching, but also as reference for unique naming within the file system tree. A graph structure does not have such a natural starting point. Two possibilities for selecting a starting point for a tree-based representation of a graph can be derived from typical usage patterns of the (classic) Web: users either are aware of a URL they want to visit (e.g., by using a bookmarking system) and navigate directly to the corresponding Web site, or use search engines to find resources that fulfil their information needs.

- *Naming.* RDF, the meta model for representation of Linked Open Data, uses URIs [4] to identify resources and the relationships between them. Per definition, URIs are globally unique, and two resources that are identified with the same URI are considered to be the same resource. In the RDF context, the inner structure of URIs is irrelevant, and a similarity in resource naming does, per se, not imply any kind of relationship between these resources.

  Naming in file systems is different: the uniqueness of file and directory names is ensured only locally, i.e., in the context of the objects' parent directory. A file's full path is unique within the local machine context

[1]Different names and semantics are used for such mechanisms in different operating systems; e.g., *alias* or *symbolic link*. Essentially all these mechanisms allow file system objects (files or directories) to virtually appear in multiple locations, i.e., they can be accessed via multiple paths.

and can be interpreted as a sequence of local names. In this regard, file systems are more restrictive than RDF, which allows for a lossless mapping from URIs to file names. However, the syntactic rules for valid URIs differ from the rules for valid file and directory names (for instance, several characters that are allowed in URIs are not allowed in file names), which must be solved by suitable escaping algorithms.

- *Literal Values.* This naming mechanism does not apply to literals. In fact, literals are more convenient and intuitive substitutes for URIs (cf. Section 3.4 of [17]), and hence it would be obvious to map literals in the same manner as resource URIs. Literals however carry an important part of information encoded in RDF: without literals, resource descriptions would consist only of a graph interrelating abstract identifiers; with literals, humans and machines are enabled to display, process, and interpret actual data about resources.

  In file systems, the actual information to be used and processed by applications is stored *within* files, and not in the structure of the file system hierarchy. This means that file-based applications are designed to read, write, interpret, and modify not directory hierarchies, but file contents. Thus it is more practical and convenient to represent RDF literal values as file content rather than to encode them as file or directory names.

- *Resource Representations.* One basic idea of the Semantic Web is that it is used to describe resources. The actual representation of a resource, however, is out of the scope of RDF since it deals only with the metadata layer. The connection between a resource's descriptions and its actual representations is usually established by *dereferencing* its URI. By doing so a client can expect to retrieve a resource's representation (in the case of information resources) or a RDF-based metadata record about a resource (in the case of non-information resources, cf. [16], Section 2.2).

  In file systems, only information resources in the sense of the Web architecture exist: the file as a conceptual entity cannot be separated from its representation. This is both an advantage and a disadvantage: on the one hand, it is possible to directly reflect a resource representation in the file system. On the other hand, a resource may have multiple representations of different types (for instance, different text formats), which (in the case of HTTP resources) clients can retrieve using content negotiation (cf. [14], Section 12). Since a file has only one (main) content stream[2], one has to find another mapping mechanism for resource representations.

Since file systems follow a relatively simple underlying model, the degrees of freedom for modeling a virtual LOD representation are limited. Summing up the issues described before we come to a number of restrictions that determine our mapping definition. The following prerequisites for our virtual file system representation of Linked Open Data sets must be considered.

1. *Resources cannot be mapped to files.* Since file systems provide a manifestation of inner structure only for directories through the containment relationship described before but not for file contents, RDF resources cannot be mapped to files, but must be mapped to directories in order to preserve their structured descriptions.

2. *Properties cannot be mapped to files.* An RDF property establishes either a relationship between two resources or between a resource and a literal string. Again, the only model element of file system that can be used to express such relationships between objects are directories and the elements they contain.

3. *Literal values should be represented inside files.* As described before, applications should be enabled to directly access literal values, but this can only be accomplished if they are represented as file contents.

4. *Resource representations should be considered.* In file systems, contents and metadata are tightly integrated, and a file cannot be considered separate from its contents. To sustain this assumption on which file-based applications are designed, it is desirable to include resource representations of various content types into the virtual file system, thus extending the scope of RDF.

5. *A meaningful root node should be defined.* For a proper file system representation, a meaningful root node should be defined that is useful both to humans and to machines. This root node should also be chosen so that it provides a permanent mapping of resource identifiers (URIs) to file paths in order to allow to maintain file paths even if the LOD set changes.

We have defined a virtual file system representation of Linked Open Data sources that considers these conditions. In the following we present this approach and discuss our prototypical implementation.

3.2 Approach

According to the considerations described before, we represent each RDF resource within a LOD set as a virtual directory (cf. Figure 1), and we collect all (known) resources within one directory called `/!resource/`. Hence each resource obtains a unique absolute path, which corresponds to the RDF principle that each resource has a globally unique URI. To determine the name of the virtual resource directory, we convert full URIs to qualified names (cf. Section 4 of [10]). We encode URI characters that are not allowed in file systems, e.g., slashes or quotation marks, using UTF-8 character encoding[3].

The representation of each resource as a virtual directory allows us to collect all information about this resource

[2]Different file systems provide mechanisms to represent multiple content streams for files; e.g., Alternate Data Streams [3], file forks [1], or extended attributes. None of these approaches, however, is easily accessible for applications and users, and also data can often not be transferred across platforms.

[3]It depends on the operating system which characters are affected by this encoding: for instance, Windows does not permit colons in file systems, whereas in UNIX-based operating systems they can be used as long as they are escaped properly.

Figure 1: Mapping of RDF resources to virtual directories

within one single point in the virtual file system, and also to uniquely refer to this resource across the entire file system.

Within the resource directory we can now represent all available information about this resource, i.e., properties that have this resource as subject. Since a property can have multiple values we represent each property as virtual directory that contains all corresponding values. This is done differently for object properties (i.e., properties whose value is a RDF resource) and datatype properties (i.e., properties whose value is a literal). For the former, we represent the property value resource as a symbolic link[4] that refers to the resource's virtual directory, as described before. This (*i*) avoids long pathnames, which otherwise would reduce the system's usability and may cause implementation problems, and (*ii*) avoids cycles and hence indefinite hierarchy depths. An example for the representation of a object property is depicted in Figure 2.

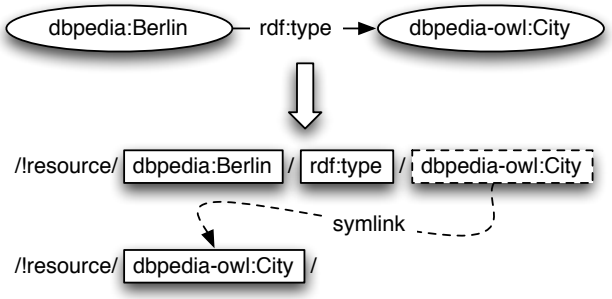

Figure 2: Mapping of object properties to virtual directories

We represent the lexical representation of datatype property values (i.e., literals) not as file name or directory name, but within the contents of a virtual file, which is located within the virtual property directory. Since a resource may have multiple properties with the same property URI but different literal values, we distinguish the single value files by numbering them (see Figure 3). This mapping provides the possibility to directly read literal strings from applications, but also to search for them using file system fulltext indices.

In addition to representing a resource's "outgoing" properties (i.e., triples that have this resource as subject) we also represent "incoming" properties (i.e., triples that have this resource as object) for convenience reasons. This representation allows a client user or application to backwards-traverse edges in the RDF graph. To distinguish incoming properties from outgoing ones, we apply the same naming convention as popular LOD browsers (e.g., Tabulator [6]) and encapsulate the property URI by `"is"` and `"of"` strings. This representation is depicted in Figure 4. Of course this mapping only applies to object properties since literals cannot be the subject of an RDF triple.

[4]A symbolic link (symlink) is a special file that contains a reference to another file or directory.

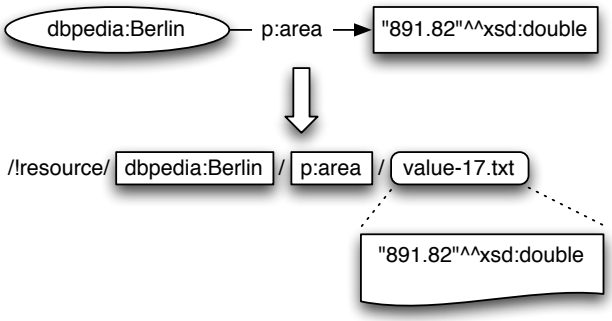

Figure 3: Mapping of datatype properties to virtual files

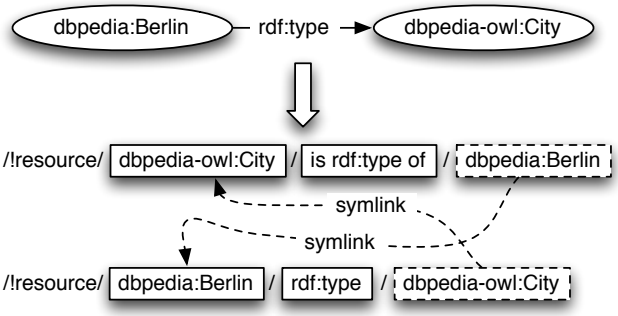

Figure 4: Mapping of incoming object properties to virtual directories

Finally, we include resource representations in our virtual file system in order to enable applications and users to directly access them without the need to deal with the HTTP protocol or other retrieval mechanisms. We represent resource contents as files that reside within the virtual resource directory, and include the representation's content type in the file name in order to distinguish them. Since the Web architecture [16] provides no means to determine which resource representations are available, we currently use three common content types (`application/rdf+xml`, `text/rdf+n3`, and `text/html`). Additionally, a comma-separated value (CSV) representation of all properties of the resource is created under the `text/csv` content type, which is of immediate use for many applications, e.g., spreadsheet tools. Figure 5 shows the resulting virtual files.

The combination of all these mappings constitutes a virtual tree-based representation of Linked Open Data sets. Using this representation, users and applications are enabled to navigate through the virtual directories that represent resources and properties, and to access property values and resource representations, which are stored as virtual files. All resources whose URIs are known can be used as starting point, since they are represented under the virtual `/!resource/` directory.

However, a LOD set may contain descriptions about large numbers of resources, and retrieving all known resources from the endpoint is an expensive task. As described in

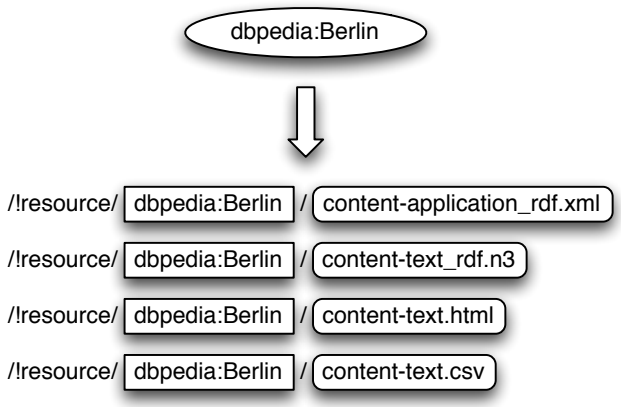

Figure 5: Mapping of resource representations to virtual files

Section 3.1 it is very common to use full text search engines as starting point for information retrieval from the Web. To provide similar behavior for linked data that is represented as virtual file system, we allow the user—in addition to the possibility of directly navigating to a virtual resource directory—to execute full text searches by creating a directory in the virtual file system's root folder, whereas the directory name is used as search term[5]. When such a folder is created, a query is issued against the LOD set and symlinks for the resulting resources are created within that directory. Such a behaviour is also implemented in a number of application-specific virtual file systems, some of which we present in Section 5.

Figure 6 shows an extract of a complete file system tree that represents data from one of the most popular LOD sources, DBpedia [2]. We can see the root directory for resources, which contains one virtual directory for each resource. Each resource contains files for representations as well as sub-directories for properties (in this example, `"p:location"` and `"rdfs:label"`), which again contain files or symlinks that represent the property values. Finally, a virtual keyword search folder (`"berlin"` in this example) is depicted that contains symlinks for each result.

3.3 Implementation

We have implemented a virtual file system driver called LODFS[6] that represents data from an arbitrary SPARQL endpoint as virtual file system. This implementation uses the FUSE-J toolkit[7] which allows for the implementation of file system drivers in the user space and thus disburdens the developer from the need to develop kernel extensions. Currently, FUSE file systems can be used on Linux, FreeBSD, and Mac OS X platforms.

A LODFS instance is always bound to one SPARQL endpoint and potentially represents all data that is available through this endpoint. When the LODFS driver is launched, it only provides a root directory that contains an empty `/!resource/` directory. The preferred way to access resources is through a full text search. Whenever a user creates a directory within the driver's root directory, a corresponding SPARQL `SELECT` query is issued, the resulting resources are added under the `/!resource/` directory, and symlinks are created within the virtual search directory. Alternatively, the user can directly access resource descriptions by creating (`mkdir`) or changing into (`cd`) the corresponding resource directory, e.g., `/!resource/dbpedia:Berlin/`.

[5]For a discussion on the practical applicability of fulltext queries in the context of Linked Open Data, refer to Section 4.1.

[6]LODFS: `http://lodfs.sourceforge.net`

[7]FUSE-J Framework: `http://fuse-j.sourceforge.net`

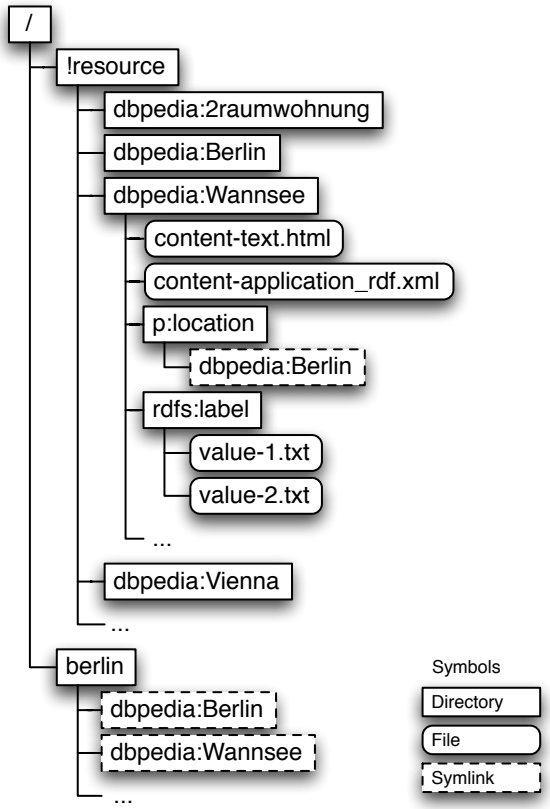

Figure 6: A complete virtual file system, representing resources from DBpedia

The implementation retrieves resource descriptions only *on demand*: when a request (e.g., a directory listing) to a virtual resource directory is issued, and the data of the resource has not yet been retrieved, a SPARQL `DESCRIBE` query is issued, and the resource representations of various types (cf. Figure 5) are retrieved. Then, the resulting resources are represented as virtual directories, files, and symlinks.

Figure 7 shows a transcript of a console session that interacts with a LOD dataset. In this example, a full text search directory is created and its contents are listed. Then, all properties and representations of one resource (`dbpedia:Wannsee`) are listed. Finally, the contents of all literal values for the resource's `rdfs:label` property are printed.

4. PRELIMINARY EXPERIENCE

So far we have discussed a number of conditions for a virtual file system representation of LOD data (Section 3.1). In Section 3.2 we have presented our mapping approach,

```
$ cd /Volumes/lodfs
$ ls
$ mkdir berlin
$ cd berlin
$ ls
0 dbpedia:Berlin@ ->
  /Volumes/lodfs/!resource/dbpedia:Berlin
0 dbpedia:Wannsee@ ->
  /Volumes/lodfs/!resource/dbpedia:Wannsee
[...]
$ ls dbpedia:Wannsee
26093 content-application.rdf+xml*
17060 content-text.csv*
30331 content-text.html*
17417 content-text.rdf+n3*
    0 foaf:depiction/
    0 foaf:img/
    0 geo:lat/
    0 geo:long/
[...]
$ cat dbpedia:Wannsee/rdfs:label/*
"Wannsee"@es
"Großer Wannsee"@nl
"Großer Wannsee"@de
"Großer Wannsee"@da
[...]
$
```

Figure 7: Transcript of a LODFS session

and in Section 3.3 a prototypical implementation of this approach was described. From the experience we have gained in the course of the design, implementation, and usage of our approach, we can observe a number of open issues in the context of LOD related technologies. In the following we outline several of these issues in order to indicate directions for further research and development.

4.1 Linked Open Data Issues

Resource Rendering.

URIs play a fundamental role in Linked Open Data, as they are used for the identification of resources and properties. Although they are not primarily designed for human consumption, URIs are also often used for the visual rendering of resources in user interfaces. A number of vocabularies provide properties designed to describe a resource's human-readable label (e.g., `rdfs:label` or `skos:prefLabel`), however their presence is not guaranteed, in which case the URI serves as fallback for rendering. Moreover, a resource may have multiple `rdfs:label` property values, or different resource's labels may be equal, which causes confusion in user interfaces.

URI Prefixes.

Long URIs are hard to render in a user interface, and they are also not directly suitable to be used as file names or directory names because of forbidden characters. In our implementation, QNames are used to abbreviate URIs with human-friendly shortcuts, and a number of URI prefixes (e.g., `rdf:` or `owl:`) can be regarded as commonly accepted. However, in principle there exists no globally valid mapping for URI prefixes since they are by definition valid only in a local context. For generic client applications like our virtual file system it is therefore hard to determine which URIs are used and which URI prefixes can be applied. URI prefixes can be embedded in the various RDF serializations (e.g., using namespaces in the RDF/XML serialization), but in practice often default prefixes are applied which have no meaning to the user (e.g., `j_0:` and similar prefixes are regularly found in RDF serializations produced by the Jena Semantic Web framework).

To overcome this drawback one could imagine metadata that describes a LOD set, and also indicates which vocabularies and URI prefixes are used therein. The recently presented Vocabulary of Interlinked Datasets (voiD)[8] includes a property (`void:vocabulary`) to describe which vocabularies are used within a dataset, but does not consider the definition of preferred URI prefixes. Thus, it would be an option to extend the voiD vocabulary towards this direction.

Another approach to solve this problem is the usage of lookup indices like the recently presented `prefix.cc` service[9], which maintains a list of mappings from prefixes to URIs. Developers can use this service to submit their prefix-to-URI mapping and to look up the full URI for a given prefix. `prefix.cc` resolves prefix naming conflicts using a voting mechanism, hence the most popular prefix mapping is determined by the user community. Currently, however, this service does not allow clients to query the preferred prefix for a given URI, which reduces its applicability for the purposes described in this paper.

Content Representation.

The Web Architecture [16] does not provide means to specify which content types can be used to retrieve a resource representation. Thus it is difficult for a generic client to identify and retrieve all existing representations. Currently, a client can only try to retrieve common content types (e.g., `text/html` or `application/rdf+xml`). A mechanism to obtain existing resource representations of specific content types would greatly increase the applicability of resource descriptions.

RDF Language Features.

A number of RDF language features (especially anonymous resources, collections, and reification) are considered problematic in the context of Linked Open Data (cf. [8], Section 2.2). Their applicability in the context of virtual file systems is also restricted, since file systems do not provide mechanisms to reflect these language elements (e.g., it is not possible to define files without a name to represent anonymous resources, or to represent reified files or directories). As it is considered good practice to avoid these features in Linked Open Data (cf. [8], Section 2.2) our approach also ignores blank nodes and treats collections and reification in the same manner as other RDF triples.

Fulltext Queries.

Currently, SPARQL provides fulltext search only through the usage of the `regex()` filter (cf. Section 11.4.13 of [21]); a typical fulltext query according to this specification is de-

[8] voiD vocabulary: `http://rdfs.org/ns/void`

[9] Namespace lookup for RDF developers: `http://prefix.cc`

picted in Figure 8. The implementation of this class of queries, however, is usually not optimal; for instance, the current DBpedia SPARQL implementation[10] runs into a timeout when this query is issued.

```
SELECT DISTINCT ?s
WHERE { ?s ?p ?o .
        FILTER regex(?o, "vienna", "i") . }
```

Figure 8: Standards-compliant SPARQL fulltext query

On the other hand, different SPARQL implementations provide fulltext search through proprietary query language extensions. For DBpedia, fulltext queries can efficiently be issued through the virtual `bif:contains` property (cf. Figure 9), which is defined by the underlying OpenLink Virtuoso implementation [12]. This query form cannot be used in a generic client since it depends on the implementation of the SPARQL endpoint, which contradicts the intention of a high-level query language; i.e., to abstract over a service's implementation specifics. It is crucial for LOD endpoints to efficiently implement a standardized mechanism for fulltext search in order to be used by generic clients.

```
SELECT DISTINCT ?s
WHERE { ?s ?p ?o .
        ?o bif:contains "vienna" . }
```

Figure 9: Fulltext queries in OpenLink Virtuoso

Updates.
Linked Open Data does not provide a mechanism to update data, hence the virtual file system is read-only. There exist proposals for an update extension to SPARQL (e.g., the SPARQL/Update proposal which is currently a W3C member submission [23]), but the "Writable Web" has been addressed only in a few number of works (e.g., [7]), and is currently being addressed also in a W3C community project[11].

4.2 File System Issues

Operating System Specifics.
There exist a number of differences regarding the file system implementations of common operating systems. For instance, the meaning of a backslash (\) in a path expression differs under Windows, where the backslash separates sub-directory names, and under Linux/Unix-based systems, where it is used to escape special characters. Even on a single platform the behaviour can be different: for instance, Mac OS X allows the usage of slashes (/) in file names, but the underlying Unix file system implementation converts them to colons. Our prototype implementation follows the convention of using a colon to separate URI prefixes from the local names; consequently these directory names show up with a slash in the Finder (cf. Figure 10). Of course, for a Windows implementation a different separator would have to be chosen.

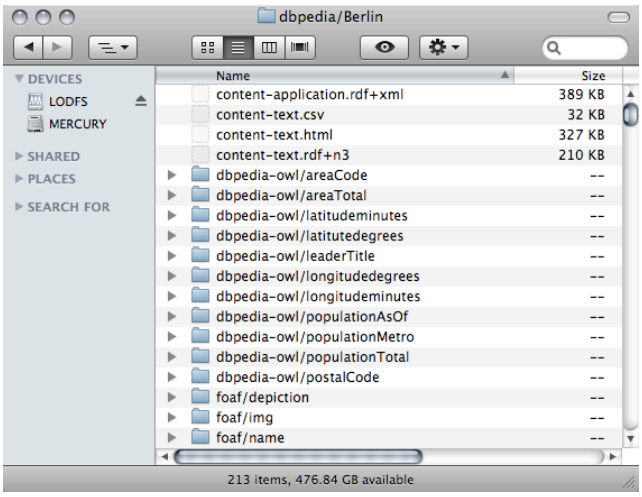

Figure 10: LOD resource representation in Mac OS X Finder

Path Lengths.
Many operating systems impose a limit on the maximum number of characters for absolute file paths. Although object properties are realized using symbolic links in our implementation, the virtual path to a resource may become very long, especially in the case of cyclic RDF properties. Currently this can be solved within applications and file browsers by resolving symbolic links.

5. RELATED WORK

The current state of the art w.r.t. the consumption of Linked Open Data for end users are RDF browsers, of which a number have been presented previously (e.g., [6, 18, 20]). These provide useful navigation interfaces for end users, but do not provide the possibility for applications to access Linked Open Data without the need to implement the corresponding client protocols or complex data transformation operations.

A number of approaches have been presented how to use (semi-)structured object annotations for the generation of virtual file system views; e.g., by interpreting file path elements as AND-combination of attribute/value pairs [11, 15], tags [9], or automatically generated classifications [13]. In this approaches the virtual file system path is translated into a query which is executed on the underlying data, and the results are presented as virtual files and sub-directories. With our virtual fulltext search directory (cf. Section 3.2) we follow a similar approach, but additionally we map each resource in the underlying data set to a fixed file system representation, which allows for permanent file path references to be made. A virtual hierarchical file system entirely built on Semantic Web technologies, which allows for additional annotations and expressive search using an extended file API, is presented in [22], and it is shown that the performance of such systems is approaching a level sufficient for interactive usage.

[10] DBpedia SPARQL endpoint: `http://dbpedia.org/sparql`
[11] pushback — Write Data Back From RDF to Non-RDF Sources: `http://esw.w3.org/topic/PushBackDataToLegacySources`

The *libferris* virtual file system [19] provides a generic architecture that allows to mount a vast number of data sources, including relational data bases, remote HTTP and FTP servers, and XML documents. Libferris provides means not only to read from these sources but also to store modifications to the virtual file system in the underlying data source (e.g., a new node in an XML document can be inserted by creating a directory in the virtual directory hierarchy), including locally stored RDF data which is accessed by the means of the Redland RDF framework[12]. RDF2FS [24] is a utility that transforms a given RDF file into an actual directory tree. Its mapping approach is comparable to the one presented in this paper, but RDF2FS is limited to locally available RDF data and does not dynamically download data from remote LOD sources. Finally, in [25] a virtual file system based on Topic Maps, which are conceptually close to RDF, is presented.

A number of approaches comparable to ours can be found for specific web applications, including flickrfs[13], GmailFS[14], or youtubefs[15]. These approaches translate file system calls to operations on the underlying service API and represent data from the service's account as virtual files. Services that deal with multimedia content like the ones described here are predestined to be represented as files since their APIs provide a unified view on content but also on annotations like tags or user comments. To the best of our knowledge, the approach presented in this paper is the first one that uses arbitrary data accessible via a SPARQL endpoint and additionally considers fulltext search and resource representation in conjunction with RDF descriptions.

6. CONCLUSIONS

In this paper we have shown how Linked Open Data sets can be represented as virtual file systems, and hence be directly used by file-based applications without further conversion steps. We have sketched a number of potential application scenarios for such an implementation, and we have discussed design considerations that influence our mapping.

Our prototypical implementation maps RDF resources to virtual directories, which contain sub-directories and files that represent object and datatype properties. We additionally include resource representations of various content types into our virtual file system in order to allow applications to directly operate on these data. From our implementation we have drawn a number of conclusions and issues that indicate how the Linked Open Data technology family can be extended and improved in order to better support generic client applications.

Currently however a virtual file system based on LOD is read-only since there exists no standardized way to modify linked datasets. We believe that controlled write access could significantly improve the applicability of Linked Open Data and related techniques, not only for virtual file systems as presented in this paper; thus we will investigate more towards this direction in the future.

[12]Redland RDF Libraries: `http://librdf.org`
[13]`http://manishrjain.googlepages.com/flickrfs`
[14]`http://richard.jones.name/google-hacks/gmail-filesystem/gmail-filesystem.html`
[15]`http://code.google.com/p/youtubefs/`

Acknowledgements

Parts of this work have been funded by FIT-IT grants 812513 and 815133 from Austrian Federal Ministry of Transport, Innovation, and Technology. The author thanks Niko Popitsch, Bernhard Haslhofer, and Stefan Zander for valuable comments on this paper.

7. REFERENCES

[1] Apple Inc. *File Forks*, 2005. Available at `http://developer.apple.com/documentation/mac/Files/Files-14.html`.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference (ISWC 2007), Busan, Korea*, 2007.

[3] Hal Berghel and Natasa Brajkovska. Wading into Alternate Data Streams. *Communications of the ACM*, 47(4):21–27, 2004.

[4] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986)*. Network Working Group, January 2005.

[5] Tim Berners-Lee. *Linked Data*. World Wide Web Consortium, 2006. Available at `http://www.w3.org/DesignIssues/LinkedData.html`, retrieved 08-Aug-2008.

[6] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, 2006.

[7] Tim Berners-Lee, J. Hollenbach, Kanghao Lu, J. Presbrey, Eric Prud'hommeaux, and m.c. schraefel. Tabulator Redux: Browsing and Writing Linked Data. In *Proceedings of the Workshop on Linked Open Data on the Web (LDOW2008)*, 2008.

[8] 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.

[9] 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.

[10] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. *Namespaces in XML (Second Edition) (W3C Recommendation 16 August 2006)*. World Wide Web Consortium, 2006. Available at `http://www.w3.org/TR/REC-xml-names/`.

[11] 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.

[12] Orri Erling and Ivan Mikhailov. RDF Support in the Virtuoso DBMS. In Sören Auer, Christian Bizer, Claudia Müller, and Anna V. Zhdanova, editors, *CSSW*, volume 113 of *LNI*, pages 59–68. GI, 2007.

[13] 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.

[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616)*. Network Working Group, 1999.

[15] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and Jr. James W. O'Toole. 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] Ian Jacobs and Norman Walsh. *Architecture of the World Wide Web, Volume One (W3C Recommendation 15 December 2004)*. World Wide Web Consortium, 2005. Available at `http://www.w3.org/TR/webarch/`.

[17] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax (W3C Recommendation 10 February 2004)*. World Wide Web Consortium, 2004.

[18] Georgi Kobilarov and Ian Dickinson. Humboldt: Exploring Linked Data. In *Proceedings of the Linked Data on the Web Workshop (LDOW2008)*, 2008.

[19] Ben Martin. The World is a libferris Filesystem. *Linux Journal*, April 2006.

[20] Eyal Oren, Renaud Delbru, and Stefan Decker. Extending Faceted Navigation for RDF Data. In *International Semantic Web Conference*, pages 559–572, 2006.

[21] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF (W3C Recommendation 15 January 2008)*. World Wide Web Consortium, 2008.

[22] 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.

[23] Andy Seaborne, Geetha Manjunath, Chris Bizer, John Breslin, Souripriya Das, Ian Davis, Steve Harris, Kingsley Idehen, Olivier Corby, Kjetil Kjernsmo, and Benjamin Nowack. *SPARQL Update – A Language for Updating RDF Graphs (W3C Member Submission 15 July 2008)*. World Wide Web Consortium, http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/, 2008. Available at `http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/`.

[24] Michael Sintek and Gunnar Aastrand Grimnes. RDF2FS – A Unix File System RDF Store. In Christian Bizer, Sören Auer, Gunnar Aastrand Grimnes, and Tom Heath, editors, *Proceedings of the 4th Workshop on Scripting for the Semantic Web*, 2008.

[25] Alexander Zangerl and Robert Barta. Virtual File System on Top of Topic Maps. In *Proceedings of the Fourth International Conference on Topic Maps Research and Applications*, 2008.