Ad-hoc File Sharing Using Linked Data Technologies

Niko Popitsch and Bernhard Schandl

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

Abstract. A large fraction of our information, both in the professional and private domains, is stored in the form of files on our personal computers. When we collaborate with co-workers or meet with friends, mechanisms for sharing files and file annotations are frequently required. However, centralized file sharing infrastructures are often not available or complicated to set up, and approaches like peer-to-peer sharing infrequently provide functionality beyond simple copying of files between machines. In this paper we present a light-weight approach for ad-hoc file sharing based on Linked Data principles. Our system exposes parts of a file system as Linked Data and allows users to interlink and annotate resources in such linked file systems. We further provide a mechanism for mounting multiple such file systems together, and for seamlessly navigating them using a common Web browser. As the exposed files and directories become Web resources, they are amenable to a large set of Semantic Web and Linked Data tools. Human and machine users may exploit such linked file systems in ad-hoc data sharing scenarios. They may further add arbitrary annotations to local and remote linked file system resources, which may also be shared among users. Finally, file system objects may be searched based on their extracted metadata and such semantic annotations.

1 Introduction

File sharing has become a central activity in the professional and private domains [1–3]. The sharing of files is supported by a large number of tools and methods, ranging from email attachments over centralized file servers to peer-to-peer sharing applications. An increasingly used method is the exchange of data via Internet-based sharing systems that may be specialized for a certain media type (e.g., Flickr, Youtube) or of general purpose (e.g., DropBox). Users select one or multiple of these tools to solve a particular sharing problem based on *what* is shared and with *whom* it is shared [1].

In this work we focus on a particular type of data sharing, *ad-hoc sharing*, which is characterized by the lack of pre-existing sharing infrastructure. Often, the participating users and their devices are physically close; however, this is not a precondition. Ad-hoc sharing is rather identified by the need to quickly exchange data with users or devices they do not often share data with (in the

past and the future) so that the setup of heavyweight sharing infrastructures is unfeasible [2].

Consider for example the following scenario: after a common vacation, Alice and Bob meet with friends to talk about their common experiences and exchange their digital photos among each other. Alice would like to give their friends a photo presentation of her and Bob's photos. Bob would like to copy some of Alice's photos to his machine but first needs to select which ones. Further, Bob would like to add information about where a particular photo was taken (e.g., what restaurant they had that great fish menu at). These annotations should be accessible also to Alice and their friends, and they should be able to extend them. Bob would further like to explicitly link related photos, e.g., he would like to relate the photos of his daughter taken during last year's vacation to this year's photos.

Although Alice, Bob, and their friends know each other well, it is rather unlikely that they exchange data frequently; therefore the introduction of a heavyweight sharing infrastructure might be immoderate. Note that this scenario does not require the actors to meet in person—everything could also be done remotely. The scenario mentioned before could partly be solved with a centralized, online data sharing application. This would, however, raise the following issues:

1. All involved devices would require Internet access, although local connectivity would be sufficient for most tasks.
2. The annotation tasks would be restricted to the functionality offered by the particular application.
3. By uploading data to an online platform, digital copies of these data are created. Annotations would refer to these copies rather than to the original files. When a user decides to manipulate a data item locally (e.g., applying a photo filter to reduce the red-eye effect), they would need to update the data manually on the online platform so that others can access this improved version.
4. Storing data on Web servers usually raises security and privacy issues.
5. Many existing sharing platforms handle only particular file types.

In this paper we present an alternative method for ad-hoc sharing based on Linked Data. We present how our filesystem, TripFS [4], can be used to expose parts of a local file system as Linked Data, and how multiple such *linked file systems* can be mounted and seamlessly navigated with a common Web browser. As the exposed files and directories become regular Web resources, they are amenable to a large set of Semantic Web and Linked Data tools. We further describe how arbitrary annotations and links can be added to such resources: resources may be linked to local and remote files exposed via TripFS, but also to any other Web resource or Linked Data source. We describe how human and machine users may exploit such linked file systems in ad-hoc data sharing scenarios as the one presented above, and conclude with a discussion of advantages and shortcomings of our approach when compared with related work from the file sharing domain.

```
1  <http://queens.mminf.univie.ac.at:9876/resource/71023c2f-8aec-41b0-ac0b-0ce38cf1e0f7>
2    a tripfs:File ;
3    rdfs:label "piran2.jpg" ;
4    tripfs:local-name "piran2.jpg" ;
5    tripfs:path "file:/g:/watch/images/2009/vacation/piran/piran2.jpg" ;
6    tripfs:size "46170"^^xsd:long ;
7    tripfs:modified "2010-07-20T10:04:59"^^xsd:dateTime ;
8    tripfs:parent
9      <http://queens.mminf.univie.ac.at:9876/resource/3bb652a5-d38c-4c01-b9b7-548c0c19e546> ;
10   nfo:hasHash "58717"^^xsd:int ;
11   nie:mimeType "image/jpg" ;
```

Fig. 1: RDF representation of a file served by TripFS. In addition to basic file system data (lines 1–7), the representation contains a triple that connects the file to its parent directory (lines 8–9) and extracted metadata (lines 10–11).

2 TripFS: Exposing File Systems as Linked Data

TripFS[1] [4] is a lightweight utility that publishes parts of a local file system as Linked Data. It bridges the gap between the distinct worlds of hierarchical file systems and the hyperlink-based Web by

1. providing *stable, de-referencable URIs* for directories and files, thereby making it possible to establish stable references to local and remote file system objects;
2. *extracting metadata* from files, thereby allowing to find and access files based on their contents instead of their location;
3. *linking files* to external Linked Data sources based on extracted metadata, thereby opening file systems for global, enterprise-wide, or personal information integration; and
4. *serving* file and directory descriptions as Linked Data (through de-referencable URIs, a SPARQL endpoint, and RDF representations), thereby providing access to file systems using standardized (Semantic) Web technologies.

TripFS combines several third-party components (including the Jena Semantic Web Framework[2], Aperture[3] for metadata extraction, the Jetty HTTP Server[4], and the DSNotify monitoring framework [5]) and can be deployed as a background process on any Java-enabled computer. It can be configured to use any RDF storage backend for storing annotations and extracted metadata. Upon start, it crawls the configured file system subtrees and builds an RDF representation where directories and files are represented as RDF resources. TripFS extracts metadata from file system objects and links these objects with each other, as well as with external data sources. After crawling, DSNotify is used to monitor changes in the file system, which are in consequence reflected in the

[1] `http://purl.org/tripfs`
[2] `http://openjena.org`
[3] `http://aperture.sourceforge.net`
[4] `http://jetty.codehaus.org`

RDF model. New or changed files are re-analyzed, so that the RDF model remains in sync with the local file system. Figure 1 shows an RDF description of a file, as served by TripFS. In addition to the RDF representation, TripFS provides a convenient HTML-based interface that allows the user to navigate through the file hierarchy. All main components of TripFS are flexible and extensible; in particular, extractors (e.g., for new file types) and linker components (for arbitrary external data sources) can be added easily.

3 Linked Data-style Ad-hoc File Sharing

In *ad-hoc file sharing*, users that do not exchange data regularly (in the past and in the future) have the short-term need to exchange file-based contents between multiple machines. As discussed, they cannot resort to permanent infrastructure (like file servers, hosting providers, or Web applications) as it is either unfeasible to set-up such an infrastructure or due to infrastructural constraints (e.g., limited connectivity, firewalls, etc.). Often, ad-hoc file sharing takes place in situations where users are co-located and have some but limited shared network infrastructure (e.g., a Wi-Fi network). Ad-hoc file sharing is of relevance both in professional and in private contexts: for instance, during a business meeting one may want to share a certain document or spreadsheet with all participants. In the private domain, one may want to exchange photos from the recent vacation with friends during a relaxed dinner.

Analyzing the related works from the file sharing domain mentioned in this paper (in particular, [2] and [6]) and combining it with our own considerations led us to the following list of requirements for ad-hoc file sharing:

1. *Universality:* all file types should be sharable.
2. *Minimum preconditions:* participants (i.e., data providers and consumers) should not require a lot of additional software to be able to share files.
3. *Minimum configuration:* setting-up a new collaborative file space should be as easy as possible.
4. *Lightweight and usable access control:* it should be simple and fast to assess shared files and to decide on access rights.
5. *Platform and network independence:* it should be possible to share files across different hardware and software (operating system) platforms. It should further be possible to share files across network boundaries.
6. *Support for transient data and stable links:* data in ad-hoc sharing scenarios is accessible only for a short amount of time. Sometimes this is sufficient in a particular sharing scenario [2]. However, sometimes operations on shared data run over multiple such ad-hoc sessions (in our case, e.g., annotations and links between files should be preserved) and sharing solutions should support such operations.

3.1 Ad-hoc File Sharing in Practice

Today, sending email attachments seems to be the predominant way of personal file sharing [3]. Voida et al. analyzed that users tend to fall-back to such a uni-

versal data sharing mechanism when they are unsure about the availability of a certain sharing tool at the recipients side, or when they have problems of communicating through firewalls [1]. Another common technique for infrastructure-less ad-hoc file sharing is to use detachable physical devices, like USB sticks. Usually, this "offline" method for file sharing works straightforward, except for limited storage capacity on the removable media. Another popular way to share files is to send them via instant messaging (IM) channels. Most of these tools provide simple mechanisms to send files to one or many chat partners, which however requires all participants to have network connectivity, an account for the IM network, and corresponding client software at hand. This method is further not applicable when the available network does not permit the usage of IM software due to security restrictions, e.g., in corporate intranets. Peer-to-peer based file sharing constitutes another often-used method [1, 3]; however, classical peer-to-peer platforms like Napster, Gnutella or KaZaA seem less applicable in ad-hoc file sharing scenarios.

Other common methods to share files make direct use of the World Wide Web, arguably one of the most important information channels today. The Web is well-supported by most modern devices: even low-capacity mobile devices allow users to access Web resources. It is easy nowadays to set-up personal Web presences without knowing the technical details of content markup and Web hosting. Because of their widespread adoption, Web technologies are a promising candidate for ad-hoc information sharing. However, current Web 2.0 applications that support file sharing suffer from the previously mentioned issues (cf. Section 1) such as the requirement for Internet access or limited annotation support. A major drawback of such centralized systems is that they require all shared files to be uploaded to their Web servers first. In our scenario, this means that Alice would have to upload all her vacation photos before Bob can select some for downloading them to his laptop. These digital objects are not directly connected with their digital "originals" residing on Alice's computer, meaning that changes to these files are not automatically propagated to the shared versions and vice versa. Further, Alice cannot directly benefit from annotations made to these online copies outside the hosting Web application itself.

In this paper we present an alternative file-sharing approach that does not require a centralized infrastructure or digital copies of resources and is based on Linked Data principles. Linked Data [7] re-uses and extends the Web infrastructure with technologies that allow to represent, transport, and access raw data over the network. In comparison to the traditional, document-centric Web it comprises the significant improvement that it associates resource identifiers (URIs) with structured descriptions that are represented in a unified format (RDF) and can be accessed by de-referencing their URIs. In the context of file systems, Linked Data techniques can be used to expose structured metadata descriptions about files, which allows clients to access them based on their semantic meaning rather than just based on their location in a file system hierarchy [4].

3.2 File Sharing with TripFS

Based on the scenario outlined in Section 1, we have extended TripFS with features that allow users to easily share files across a (local) network, and to connect multiple file systems using Linked Data technologies. In the following we reconsider the scenario and describe which particular TripFS features support this use case.

One-click Sharing. When Alice and Bob meet to discuss and exchange their recent photos, both want to share folders (including subfolders) on their laptops that contain these photos[5]. When Alice starts TripFS on her laptop, it announces its service URL via a Zeroconf[6] service, so that it can be discovered by other machines on the same network. In parallel, TripFS crawls the selected part of the local file system, extracts metadata from files, and links them to other data sources (cf. Section 2). The resulting triples are incrementally stored in the RDF store and are immediately published via the Linked Data interface.

For adding new directory subtrees to TripFS, Alice makes use of the TripFS Windows Explorer shell extension[7] that allows to share a folder with a single mouse click (cf. Figure 2). When Alice clicks this button, a local Windows socket is opened and the selected directory's path is sent to TripFS. TripFS adds this directory to its list of exposed root directories and creates a new observed region for DSNotify. The shell extension reports the successful or unsuccessful outcome of this operation to the user via a popup dialog. Immediately, the folder is accessible via the Web server built into TripFS and can be accessed by devices on the network. If the newly exposed root directory lies within a directory that is already exposed, TripFS marks it as inactive in order to avoid unnecessary monitoring and crawling costs for overlapping regions. For the same reasons, TripFS deactivates all existing root directories that lie in a subtree of a newly added root directory.

Accessing Shared File Systems. Since TripFS provides both, an HTML- and an RDF-based view on shared folders, Alice's friends can access her photos using the Web browser installed on their laptops. If their system supports service discovery via Zeroconf they not even have to enter the hostname or IP address of Alice's laptop. They can navigate through the file hierarchy and download their favorite photos (a screenshot of this interface is presented in Figure 3). They could also use the structured data exposed by TripFS to search for files using a visual Linked Data query builder (like, e.g., *Explorator* [8]), which allows to visually construct structured queries. For example, Bob may decide to download only photos taken on a certain day (indicated by EXIF metadata extracted from the photos), or photos that are related to a particular place (represented by

[5] Let us assume they have access to a shared wireless network.
[6] Zero Configuration Networking (Zeroconf): `http://www.zeroconf.org`
[7] It is also possible to add shared directory subtrees via the Web interface.

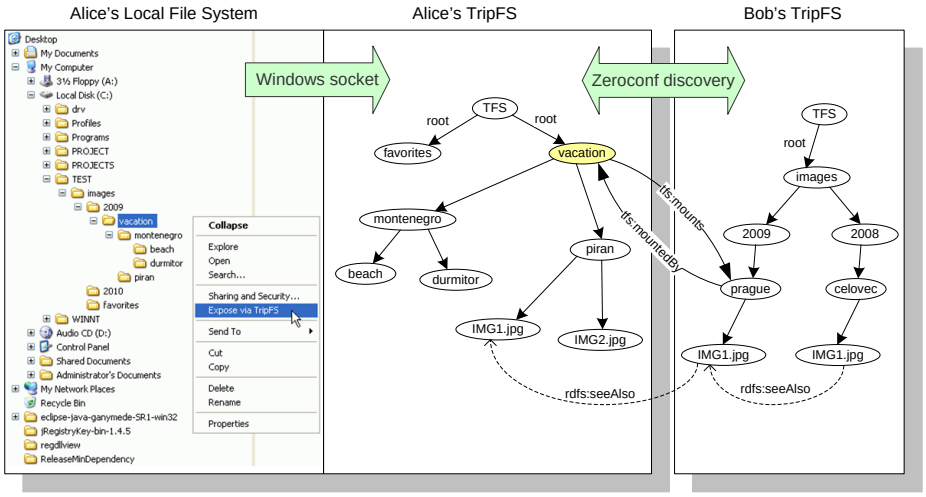

Fig. 2: Ad-hoc file sharing with TripFS: Parts of Alice's file system depicted on the left have been exposed by her TripFS instance, depicted in the middle. The screenshot on the left shows the Windows explorer shell extension for one-click sharing of file resources via TripFS. The "vacation" resource is the mount point of Bob's (remote) TripFS. The dashed arrows denote explicit links between files.

a link to a GeoNames entity that has been created based on extracted GPS coordinates).

Annotating Shared Files. While he browses Alice's photos, Bob wants to annotate one of the pictures because he remembers the particular restaurant where the picture was taken. For this purpose, TripFS provides an *RDF sink*. This component establishes a Web resource that accepts RDF data (for instance, annotations of shared files) sent by clients via HTTP POST, and stores posted triples in a designated named graph within the TripFS RDF store. Later, these annotations are published together with metadata that have been extracted from files. For instance, Bob could drag the URL of the restaurant's Web page from his Web browser to a designated area on the TripFS HTML interface, causing an `rdfs:seeAlso` triple to be stored. If Bob did this for multiple files, he could later retrieve all photos linked to the restaurant's Web page through a structured query, as described before.

Mounting Other TripFS Instances. A special form of file annotation is *Linked File System Mounting*. This technique uses Linked Data principles to connect distributed file systems, similar to the well-known *mount* operation in UNIX-like operating systems (cf. Figure 2). TripFS defines an RDF property

`tripfs:mounts`[8] to link a directory in one instance to a directory in the same or another one. TripFS provides a simple user interface for mounting remote TripFS instances, which leads to the creation of the respective triples in both involved TripFS RDF stores. Applications may add mount links by simply posting a respective triple to the RDF sink. Mount triples should be interpreted by TripFS consumers (such as the Web-based TripFS file browser) like parent-child relationships to enable seamless navigation across file system boundaries. Note that in contrast to local file systems, it is possible to create circular structures using Linked File System Mounting. Although the TripFS RDF sink rejects mount triples that would directly lead to such a situation, circles cannot be generally avoided due to the distributed, open-world nature of Linked Data. Consumers (e.g., crawlers or user interfaces) need to be aware of this possibility to avoid unwished complications like infinite loops. As mount triples reside in the annotation model of the TripFS instance they were posted to, a mount link is initially visible only to clients of this particular instance, as it is the case with UNIX mounts. However, following the idea of the Web of Data, it is reasonable to propagate the mount triple also to the remote TripFS instance, so that it can be easily followed backwards. Thus the RDF sink posts a respective `tripfs:mountedBy` triple to the RDF sink of the remote TripFS. Since TripFS provides stable URIs for files due to its file-monitoring component, these mount points remain valid even if a mounted file system is temporarily unavailable, or if the user decides to move a shared directory to a different location on their hard disk.

Seamless File Browsing. While TripFS allows Alice and Bob to interlink their file systems and mutually add annotations to exposed files, this environment still provides no seamless browsing experience: for instance, file annotations are exposed only by the TripFS instance they are stored in. However, if Bob wants to add a private annotation to one of Alice's files, it should not be stored in Alice's TripFS instance but in Bob's, and he wants this private annotation to appear when he browses Alice's file system.

To overcome this issue, TripFS contains a Web-based proxy browser that dynamically fetches RDF descriptions from remote sources and enriches them with annotations from the local TripFS RDF store (cf. Figure 3). Annotations are stored in a separate RDF graph in TripFS that is merged with a resource's (remote or local) RDF graph for rendering purposes. Thus, annotations from the local store that refer (via their subject URI) to resources in the remote source are automatically mashed with the remote source's RDF descriptions: the user is presented with a single, comprehensive view of remote and local resources.

Duplicate Detection. TripFS provides a simple solution for the detection of duplicate files across multiple file systems. For each published file, TripFS calcu-

[8] `tripfs:mounts` is a sub-property of the `tripfs:child` property, an inverse property `tripfs:mountedBy` is available. The current version of the TripFS vocabulary is available at `http://purl.org/tripfs/2010/06`.

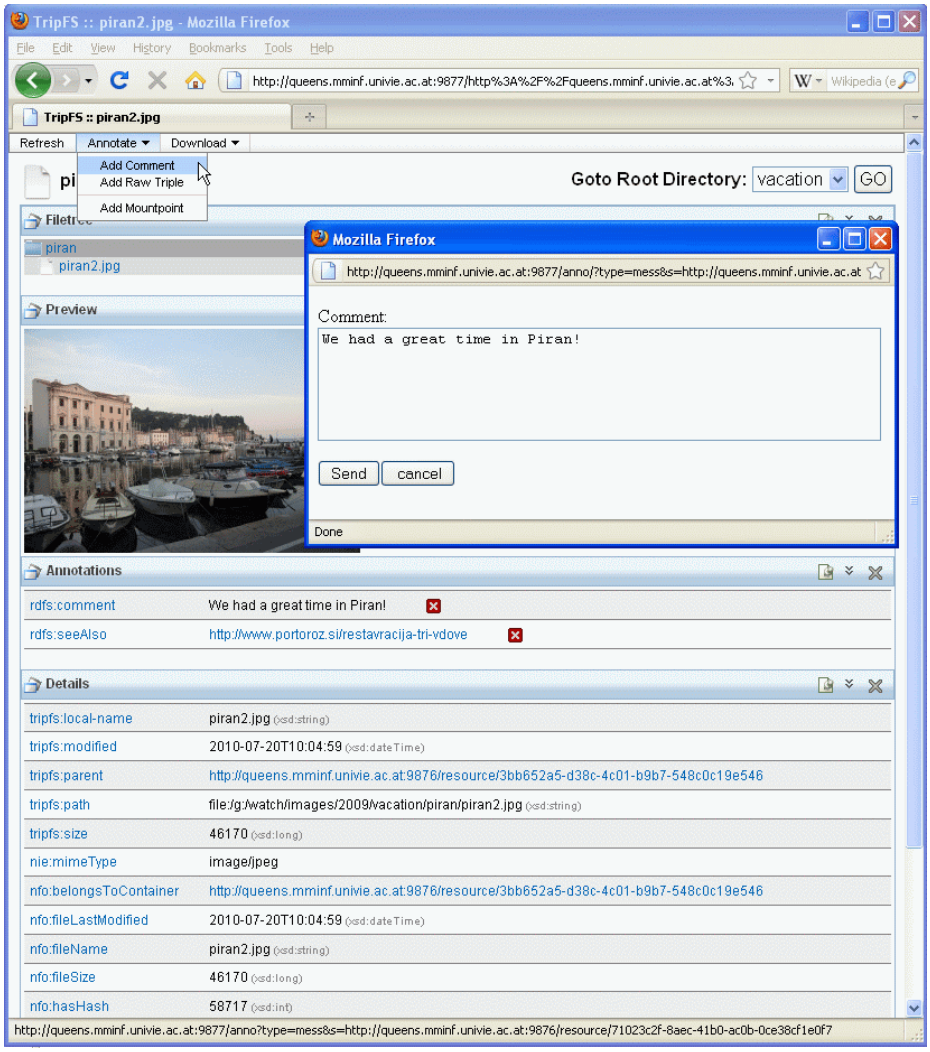

Fig. 3: The Web-based TripFS file browser. This locally running Web application can retrieve local and remote TripFS descriptions and renders them together with annotations retrieved from a local RDF model. Annotations can be added by posting RDF graphs to a servlet or via the Web interface.

lates a content-based checksum and publishes it as property of the file resource. A linker component creates `owl:sameAs` links between files within the published file system, as well as files in other TripFS instances that share a common checksum. For example, when Alice's TripFS is discovered by Bob's TripFS (and vice versa), this linker component is activated and creates `owl:sameAs` links between all duplicates found in Alice's and in Bob's shared folders. By this, Bob is enabled to immediately detect that he already copied a certain file from Alice's laptop last time they met. Further, these `owl:sameAs` links can be exploited to access resource copies when the originals are currently not accessible.

Discovery. Currently, TripFS makes use of a Zeroconf service to discover other TripFS instances. When a new instance is discovered, the duplicate detection linker described above is activated and files with equal checksums are interlinked. One drawback of the current solution is that it is restricted to the local subnet. An alternative method would be to use URNs (e.g., PURLs) for locating physical TripFS addresses. Once the PURL of a particular TripFS instance is known (e.g., because it has been communicated via email), it would remain stable. Disadvantages would be that access to the URN service would be required, and that users have to notify these services whenever their physical address changes (e.g., due to a newly assigned IP address). However, this last step can be easily automated. Another possibility is that the creation of a guest account for a particular TripFS (see below) results in an email that sends an appropriate link to a set of recipients. This link would contain the respective TripFS location as well as the required user credentials for accessing it.

Access Control. The willingness to share data with others often depends on whom these data will be shared with [9]. Access control mechanisms are therefore required also in ad-hoc sharing scenarios. As TripFS is still in a prototypical phase and as security was not our primary research goal, we have no yet implemented access control mechanisms. However, TripFS provides an increased level of privacy and security compared to other sharing platforms since the data remains under full control of the user and is not replicated to external servers.

We are however aware that *usable access control* mechanisms are essential for a system like ours. A first, straightforward solution would be to expose files via HTTPS and introduce password protection, which can be based on the underlying operating system's authentication and permission system. TripFS therefore could reuse already existing mechanisms and would avoid the need to maintain parallel structures. Additionally, a TripFS instance owner could create a new *guest* account that would be valid for a limited time with a single mouse click. The respective credentials could then be transferred to the TripFS consumer via out-of-band methods (e.g., via email or phone). Although this might be sufficient in the discussed ad-hoc sharing scenarios, more fine-grained access control mechanisms and access rights, as discussed for example in [9] and [1], have to be considered in the future.

4 Related Work

Several studies on personal file sharing focused on particular file types (e.g., music or photographs [10, 11]) or on collaboration in corporate intranets.

In [1], the authors analyze several tools and methods for data sharing and report on dimensions for characterizing them. For example, they distinguish between push- and pull-oriented systems and present a user interface for their own peer-to-peer file sharing infrastructure. In accordance with the terminology of that paper, TripFS would be a pull-oriented system that supports public or selective addressing (when password protected) and supports notifications via the DSNotify event log mechanisms [5]. The location of the files during sharing remains the provider's machine.

In [6], Rode et al. identify four significant requirements for their own ad-hoc peer-to-peer file sharing software: (i) zero-configuration for setting up a collaborative file space, (ii) no prior registration of participants required, (iii) no restriction to a fixed infrastructure (e.g., Internet access) and (iv) platform independence. We believe that TripFS meets all these requirements, although the TripFS software has to be installed on all machines that expose their files.

In [2], Dalal et al. identify a number of key problems that are not properly addressed by current data sharing technologies. The authors describe the requirement for *ad-hoc guesting*, where users require transitory, lightweight solutions for sharing data securely with unplanned sets of people with whom they have not previously shared data and that can possibly not be addressed by traditional access control. Similar to Rode et al., they identify minimal setup effort and no need for *a priori* preparations by the participants as key requirements for ad-hoc sharing. Additionally, they encourage the use of universal identifiers (e.g., email addresses) for the identification of users.

5 Conclusions

In this paper we described the current state of TripFS and its extensions since our last publication [4], namely: one-click sharing support; mounting support; seamless file browsing support across distributed, mounted linked file systems; annotation of file system objects and duplicate detection.

We further presented how this *linked file system* can be used in ad-hoc file sharing scenarios. Matching our system to the requirements described in Section 3 we can state that TripFS can be used as a universal file sharing tool that is not restricted to particular file types. TripFS requires no *a-priori* preparations for recipients of shared data. Users that actively share the data need a local TripFS instance that can either be started automatically by the operating system or by double-clicking a JAR file. TripFS allows remote users to browse the shared contents directly on the remote machine before downloading (subsets of) it. It thereby comprises a pull-oriented sharing strategy [1] that is not based on centralized infrastructure like current Web 2.0 applications. Sharing a directory subtree with TripFS is made easy by its Windows shell extension, and we consider the development of comparable tools for other operating systems.

TripFS has been implemented in Java and can be used on all platforms that support Java 1.6 or higher. In the future we aim to explore how TripFS can be deployed on mobile devices like cell phones, which are presumably more often involved in ad-hoc sharing situations. Then, TripFS could additionally be useful in "*sharing with myself*" scenarios [2], e.g., for copying photos from a person's cell phone to a desktop computer or vice versa.

References

1. Stephen Voida, W. Keith Edwards, Mark W. Newman, Rebecca E. Grinter, and Nicolas Ducheneaut. Share and Share Alike: Exploring the User Interface Affordances of File Sharing. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 221–230, New York, NY, USA, 2006. ACM.
2. Brinda Dalal, Les Nelson, Diana Smetters, Nathaniel Good, and Ame Elliot. Ad-hoc Guesting: When Exceptions are the Rule. In *UPSEC'08: Proceedings of the 1st Conference on Usability, Psychology, and Security*, pages 1–5, Berkeley, CA, USA, 2008. USENIX Association.
3. Tara Whalen, Elaine Toms, and James Blustein. File Sharing and Group Information Management. In *Personal Information Management: PIM 2008*, 2008.
4. 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.
5. Niko Popitsch and Bernhard Haslhofer. DSNotify: Handling Broken Links in the Web of Data. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 761–770, New York, NY, USA, 2010. ACM.
6. Jennifer Rode, Carolina Johansson, Paul DiGioia, Roberto Silva Filho, Kari Nies, David H. Nguyen, Jie Ren, Paul Dourish, and David Redmiles. Seeing Further: Extending Visualization as a Basis for Usable Security. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 145–155, New York, NY, USA, 2006. ACM.
7. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data — The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3), 2009.
8. Samur Araujo and Daniel Schwabe. Explorator: A Tool for Exploring RDF Data Through Direct Manipulation. In *Proceedings of the 2nd International Workshop on Linked Data on the Web (LDOW), Madrid, Spain*, 2009.
9. Judith S. Olson, Jonathan Grudin, and Eric Horvitz. A Study of Preferences for Sharing and Privacy. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1985–1988, New York, NY, USA, 2005. ACM.
10. Barry Brown, Abigail J. Sellen, and Erik Geelhoed. Music Sharing as a Computer Supported Collaborative Application. In *ECSCW'01: Proceedings of the seventh conference on European Conference on Computer Supported Cooperative Work*, pages 179–198, Norwell, MA, USA, 2001. Kluwer Academic Publishers.
11. Andrew D. Miller and W. Keith Edwards. Give and Take: A Study of Consumer Photo-sharing Culture and Practice. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 347–356, New York, NY, USA, 2007. ACM.