TagFS: Bringing Semantic Metadata to the Filesystem

Simon Schenk, Olaf Görlitz, Steffen Staab
Institute for Computer Science
University of Koblenz
http://isweb.uni-koblenz.de
{sschenk, goerlitz, staab}@uni-koblenz.de

1. INTRODUCTION

Tagging has recently become very popular because of internet applications like del.icio.us and flickr which allow easy categorisation of personal information plus sharing it with a large community. These tools are centralised internet services enabling users to collaborate, organise and share personal information. Most tagging applications are tailored to a specific set of information objects that the user manages online at a centralised storage site. To push tagging towards becoming a significant part of user's everyday work it should be integrated in a broad range of desktop applications. Today the tool most commonly used for structuring knowledge among average users is the filesystem. In the following we introduce TagFS which allows tagging of files as well as tag-based browsing for arbitrary information objects on top of the local filesystem. Tagging information is stored in RDF in order to enable easy integration with semantic web and semantic desktop applications.

As a use case, attending a conference is a scenario in which many information objects become relevant: photos taken at the conference, electronic tickets and reservations, electronic papers, etc. However, when surveying latest photo shots for sharing on a photo server, when compiling the latest travel cost statements, or when sorting the papers to be read by colleagues, hierarchical organisation of information objects is inconvenient. In contrast, tags allow for structuring an information object into the different dimensions for which it is relevant.

Keywords

Filesystem management, semantic desktop, tagging

2. ARCHITECTURE

Representing information about tagging in an ontology has the advantage that extensions of the data model and integration with other semantic aware applications are easy to realise. Figure 1 depicts the ontology used for TagFS.

Demos and Posters of the *3rd European Semantic Web Conference (ESWC 2006)*, Budva, Montenegro, 11th - 14th June, 2006

TagFS[1] provides filesystem operations (list directory, create directory, create file, delete file, etc.) that let legacy applications work seamlessly with TagFS while new applications can utilise the full power of the tagging-based infrastructure through extended interfaces on top of the metadata store. Though TagFS provides all the usual filesystem operations, the semantics of these operations have been changed significantly.

2.1 Metadata and Views

TagFS manages all filesystem information as metadata in a RDF repository following our tagging ontology. All actual files are stored in a file repository, currently an underlying conventional filesystem, using unique IDs internal to TagFS. The metadata-based approach allows for large flexibility. In particular, it allows to treat other information objects, such as bookmarks, addresses or emails, equally like files.

The semantics of filesystem operations are defined by queries and update operations on the metadata, i.e. the RDF graph, plus some minor bookkeeping for physical storage. We also define *views* that translate into SPARQL queries. A view, i.e. the corresponding SPARQL query, is applied on a RDF graph and always results in another RDF graph allowing for functional composition.

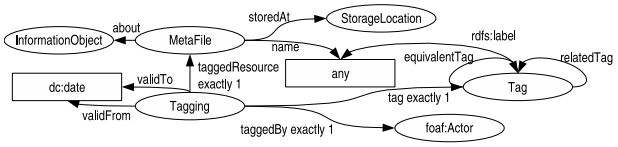

Figure 1: The Tagging Ontology

2.2 Working Directory vs. Context

An important view, called 'hasTag' was defined to select files related to a given tag. For example, the view hasTag(/, 'paper') returns from the complete metadata repository (denoted by '/') all MetaFile identifiers tagged by 'paper' and their associated data. Similarly, hasTag(hasTag(/, 'paper'), 'WWW2006') composes two views returning all MetaFile identifiers tagged 'paper' and 'WWW2006'.

We provide a shorthand query notation for the 'hasTag' view and its composition, e.g. '/paper/WWW2006' for the running example, being equivalent to '/WWW2006/paper', because the composition of hasTag-views is commutative.

[1]This research was partially supported by the European Commission under contract FP6-027026, Knowledge Space of semantic inference for automatic annotation and retrieval of multimedia content - K-Space. The expressed content is the view of the authors but not necessarily the view of the K-Space project.

Obviously, the shorthand syntax for hasTag-views shows many correspondences with common directory names. When a legacy application changes a current working directory, e.g. from '/' to '/a/b/c', the semantics of subsequent filesystem operations ('ls', 'rm *', etc.) will be defined as being executed on the RDF graph returned by the query '/a/b/c' rather than on the complete repository denoted by '/'. We will call the result of a query like '/a/b/c' which is acting as a kind of "working directory" a *working context* or simply *context* and the query itself the *context description*.

Hence, the specification of a directory like '/a/b/c' becomes a complex metadata graph instead of a simple node (i.e. directory) in a tree with subnodes (i.e. subdirectories). To simulate the listing of directories (ls), we provide another view, called LS, with signature LS: graph → list returning a list of file- and subdirectory names. This is neccessary to map view contents to the flat representation required by the common filesystem interface.

2.3 Modifying Filesystem Information

The set of possible *modification operations* on the metadata repository is described in Table 1 with *ctxt* being a context and *metafile* being the involved MetaFile identifier.

`addProperties (ctxt, metafile)`	Add statements to the metadata graph so that metafile is included in context ctxt.
`addProperties (ctxt1, ctxt2)`	For all MetaFiles `metafile` in context ctxt2 `addProperties(ctxt1, metafile)`.
`removeProperties (ctxt, metafile)`	Remove statements from the metadata graph so that metafile is no longer in context ctxt.
`removeProperties (ctxt)`	For all MetaFiles `metafile` included in context ctxt `removeProperties(ctxt, metafile)`.

Table 1: Repository Operations

2.4 Mapping Filesystem Operations to TagFS

Table 2 summarizes the mapping of filesystem operations to repository operations. Context descriptions have the form of filesystem paths. If no context description is explicitly passed as a parameter, we assume that it is implicitly given by the path to the working directory. When calling an operation for modifying metadata, contexts are resolved from context descriptions by executing the corresponding query.

The reader may note that only operations like *read, write, create, copy* need to distinguish whether the referenced object was a proper file or rather another kind of information object, such as a bookmark or address. To achieve this distinction, these operations are delegated to *ClassHandlers*, which implement them specifically for a certain class of information objects. For local files, the operations are then forwarded to the underlying storage system (in our case the underlying legacy file system).

3. IMPLEMENTATION

Our Linux-based implementation uses fuse[2] and fuse-j[3] which provide access to the Linux filesystem API from userspace and expose corresponding java-bindings. Sesame 2.0[4] is used as RDF repository. Views are not implemented as simple queries but as objects with a method taking a graph and additional parameters and returning a graph. The view object also provides a method which, given a graph, view parameters and a file returns a graph containing all statements which are neccessary for the File to appear in the context described by the parameters.

`move oldCtxdesc/File newCtxdesc/File`	removeProperties(oldCtxdesc, MetaFile); addProperties(newCtxdesc, MetaFile)
`rename File newFile`	removeProperties(ctxdesc, MetaFile); create new metafile with new file name; addProperties(ctxdesc, newMetaFile)
`delete File`	removeProperties(ctxdesc, MetaFile)
`create subdirectory`	addProperties(subdirectory, placeholder[1])
`rename oldCtxdesc newCtxdesc`	addProperties(newCtxdesc, oldCtxdesc); removeProperties(oldCtxdesc); make sure not to remove statements in the intersection of old and new context.
`delete ctxdesc`	removeProperties(ctxdesc)
`link File newCtxdesc`	addProperties(newContext, MetaFile)
`link File newCtxdesc/newFile`	Create new meta file referencing the same information object as old MetaFile; addProperties(newCtxdesc, newMetaFile).
`link ctxdesc1 ctxdesc2`	addProperties(ctxdesc2, ctxdesc1)
`create File`	Create a new information object referenced by MetaFile, addProperties(ctxdesc, MetaFile)
`read File`	Read from referenced information object
`write File`	Write to referenced information object
`copy File`	like create file followed by a write

Table 2: Mapping Filesystem Operations to TagFS

The view hasTag returns a subgraph containing all `MetaFiles` which have a `Tagging` relation to the tag with the label given as parameter, except a tagging has a validTo property which refers to a time in the past. Additionally, all attributes and `Tagging` relations of the matched `MetaFiles` are included in the result graph. `addProperty` returns a graph containing taggings, which also include taggedBy and validFrom properties. `removeProperty` sets the validTo property.

Only those classes of information objects, for which class handlers exist, are displayed: By the time of writing only a class handler for local files has been implemented. In addition to TagFS we have implemented a filesystem crawler which tags all files from a given path with tags derived from their directory and file names. This reduces the coldstart problem of not having any tagged resources and makes tagging of new files very easy. Files created later within this directory tree are automatically tagged in the same way.

4. CONCLUSION

We introduced TagFS, a tagging filesystem capable of tagging arbitrary information objects. Future development will focus on feature enrichment, integration with semantic desktop applications and tag dependency analysis based on occurrences and use patterns.

First we will improve the handling of information objects other than physical files and develop views on the history of taggings. We plan to integrate tagFS with Gnowsis[5], a semantic desktop environment. A major drawback of the flat tag space is its size, which can easily comprise some 100 tags. Hence, an important feature is tag clustering in order to reduce the number of tags displayed in one directory to improve usability. Intelligent clustering algorithms could make use of usage statistics of tags and of the relations between tags e.g. through conceptual clustering.

[2] `http://fuse.sourceforge.net/`
[3] `http://www.select-tech.si/fuse/`
[4] `http://www.openrdf.org/`
[5] `http://www.gnowsis.org/`