A Storage Ontology for Hierarchical Storage Management Systems

Sandro Schmidt, Torsten Wauer, Ronny Fritzsche, and Klaus Meißner

University of Dresden, 01062 Dresden, Germany
{sandro.schmidt, torsten.wauer, ronny.fritzsche, klaus.meissner}@tu-dresden.de

Abstract. The increasing capacity of storage media could store a huge amount of data while the price per Gigabyte is decreasing. On the other hand users and companies produce much more information that still overruns the available storage capacities. As a consequence, the management and storage of this data is very complex and expensive. Users and apps want to access their files directly and without any time delay, resulting in using fast but very expensive storage media. A deeper look on the usage of data, especially in companies, shows that only a very small amount of it is used every day. Summarizing these facts, a concept is needed to find out which data is important to provide them on fastest storage media, and less important one on cheapest storage media. Concepts derived from the Semantic Desktop can be a solution. We introduce a concept to describe files by an ontology. This allows for the description of their relations to each other, as well as their attributes. The resulting ontology offers an extensive volume of data that helps to find the adequate storage conditions for every single file. Another great advantage that showed up, is the independence from file system accesses to gather information about the stored files.

Keywords: Semantic Storage, Ontology, Hierarchical Storage Management

1 Introduction

As Gantz described in his IDC White Paper [4], there is a huge amount of information that overruns the available amount of storage capacity. That does not mean that all of this information represented as files is important during the whole lifetime. Studies showed that only 1 % of all files stored in a file system with 200,000 files are modified daily [15]. Furthermore, some data are more important, oriented on their content than others, or some should only be stored on special storage media. Some files are duplicates or redundant. As a consequence, only

Acknowledgments - Parts of this paper have been researched within the scope of the SENSE project which is funded by the Federal Ministry of Education and Research, German Aerospace Center. It is part of the *KMU-Innovativ: IKT* campaign and goes by the funding number FKZ 01IS11025D.

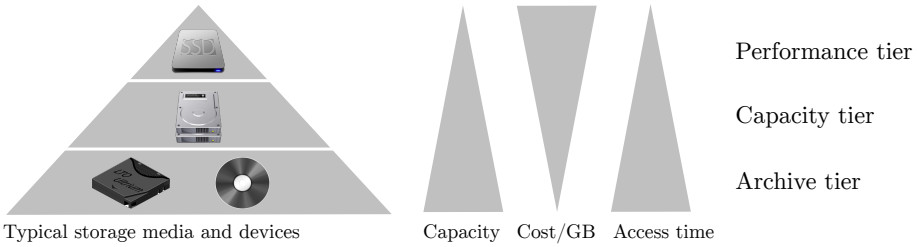

Fig. 1. Comparison of the three typical tiers of a HSM

a fraction of the stored files in a system needs to be on fast and, therefore, expensive storage media like the newest solid state drives to provide them in a performant way to users or programs.

Hierarchical Storage Management (HSM) Systems solve this problem by storing files on the most efficient storage media respective to the importance of these files. Efficient means the balance between the access frequency, the time it needs to load files from slower storage media, and how expensive the storage is. Thus, files that have a high importance because, e. g. , they were accessed a lot, should be accessed very fast by users or programs. Therefore, it is necessary to store these files on media with low access time and high data rate. Conversely files that are used rarely and, therefore, seems to be needed less often in future, should be stored on cheap media.

As shown in Fig. 1, a classical HSM System is divided into three tiers: Performance Tier (PT), Capacity Tier (CT) and Archive Tier (AT) [12, 6]. The PT has the fastest but most expensive storage media (e. g. , solid state drives), while the AT utilizes the cheapest but slowest storage media like Tape Libraries (Fig. 1). The CT acts as a compromise between cost and performance and uses storage media like SATA RAID systems, since they are faster than tape storage media but a lot cheaper than solid state drives. Due to economical reasons, the PT has the lowest storage capacity and the AT the greatest one.

Before the initialization phase no user, program or other data is stored on the HSM System and all new files will be placed down on the PT. If there is no more capacity available on the PT, files were migrated to the next lower tier and the same happens when there is no more capacity available on the CT. In practice, this migration is done in cyclic jobs, e. g. , once a day. The difficulty now, is to find the right files to migrate, since the migration is expensive due to read and write actions on the storage media and one does not want to migrate files that one need often in the future. A trivial method is to find all files that had no access for a specific time. Furthermore, every file would be analysed without context and relationship to other files. This reveals the main problems of classical HSM Systems. There is a lack of semantic and the very restricted amount of values to decide if a file has to be migrated between tiers. The lack of semantic means, that the system can not analyze files and consider files that are related by the same topic as a group of files and should be migrated together. A

second problem is the very restricted amount of file attributes such as file size and last access time.

The next section introduces our Storage Ontology (SO), that is the main part to overcome the lack of missing relations between files (Section 2). In Section 3, we present our experience and results we have made with a practical realization of our approach. Closing with related work (Section 4), we summarize our results and give an outlook in Section 5.

2 Storage Ontology

The purpose of the Storage Ontology (SO) is to offer an optimal information repository to find files that should be migrated from one tier to another. In this paper, we want to restrict to HSM Systems as described in Section 1. We also limit the files to be stored on the HSM to documents created by programs and users. This excludes files belonging to the operating system. The domain of the SO deals with files on the one side and HSM Systems on the other side and, of course, the interaction between both. This involves every information that is necessary to determine the best place of storage for each file. The following subsections provides a deeper look into the domain of the SO, starting with the domain file and going on with HSM.

2.1 Analyzing the Domain

Imagine a typical HSM System described in Section 1. Each of the files stored on it, are going to be analysed, and the extracted information is stored in a repository. Our purpose is not to collect each single possible information, since this ends up in a repository too big, where querying is not much faster than on file system level. Using too little semantic information leads to lack of knowledge and one can not determine the correct files for migration. To find a trade-off for this conflict, we took some typical scenarios from the real world to analyze which information is important. Starting with analysing files, we have a closer look on what is special about files that come from the same raw material, have the same project, or files that are grouped by the same topic.

Same raw material: At first, we observed that a lot of files are generated from the *same raw material*. E. g., professional cameras are taking photos in a very rich raw format, that needs a lot of storage capacity. After importing them, a graphic designer edits the photos and exports them maybe as JPEG files, which need less storage capacity. Later, these photos are presented on a website and small teasers from these photos are created. In this case, we don't need the raw photos from the camera any more, since we have more performant photos and the teaser ones.

This small scenario shows the following. First, there is a *group* of files that comes from a specific *source* (camera) at a specific *time*. Second, there are three *different characteristics* resulting from the *same raw material*, having different

amount of information. Third, there is a *purpose of use* for each of the characteristics. Among others, one could decide about the storage medium from this information, e. g. , the preview photos should always be stored in the PT.

Same project: As aforementioned, files often appear in groups. Groups are very important for the migration, because they offer the possibility to migrate more than just one file with only one query. As an example, imagine a video project. It consists of lots of files such as material from a video camera or audio files that should be underlayed into the video. There are also different chapters that occurs in different files. Also, the finished film, the trailer or specific still frames e. g. , for the web, could be contained in this project. As a consequence, if lots of files from one project are accessed permanently, one could infer that none of the files should be migrated to lower tiers. Only if the project is finished and very few accesses are made on the project files, these could be migrated in one step.

Same topic: Imagine there are lots of Christmas photos stored on the HSM System. Assume minimal accesses are made on these photos in the middle of the year, these photos are stored on the archive tier. But when Christmas time is coming, photos related to this topic are accessed more than in the middle of the year. Related to the migration of the files belonging the same topic, one could easily archive photos from this group in the archive tier or put them back on the higher tiers when a special topic becomes more important. Such topics could be inferred from more accesses on files related to this topic.

In every scenario, information related to time and place of a file is also needed, if the important files for the migration should be found. Therefore, it's necessary to know which storage media could offer the storage conditions that are needed for a special file. As introduced in Section 1, a typical HSM System is build up into three *Storage Tiers*, the *Performance Tier*, *Capacity Tier* and *Archive Tier*. Some of these tiers have directly mounted *Storage Media*, like solid state drives, while others have mounted whole *storage devices*, like tape libraries, that consists of several storage media (tapes). As defined, devices do not directly store files, they are more of an overlaying construct that is necessary to read, write and manage their containing media. Only physical storage media can directly store files and we do not want to lose the possibility of knowing where a file is stored concretely. Furthermore, an HSM System could consist of several *Storage Vaults*. Each of them manages one tiered hierarchy. Fig. 2 shows the structure of our understanding of an HSM System with the help of Extended Backus–Naur Form (EBNF). Since we map these structure into an ontology, EBNF is a good, simple and formal tool. Additionally, every Storage Medium should be characterized by its typical values like read/write access, average throughput or access time.

Furthermore, it is important to know about typical properties of every storage device and medium such as storage capacity, access time or data rate, while the latter could be distinguished into read and write values. These values are important to decide on which storage tier the devices or media should be placed, even in future when faster media is available and the current fast ones are not sufficient for the performance tier.

```
HSM              = { Storage Vault }
Storage Vault    = Performance Tier, Capacity Tier, Archive Tier
Performance Tier = Storage Tier (* also for Capacity and Archive Tier *)
Storage Tier     = { Storage Device }
Storage Device   = { Storage Media }
Storage Media    = { File }
```

Fig. 2. Domain of HSM described in Extended Backus–Naur Form (EBNF)

2.2 Modeling of the SO

Ontologies are formal and their purpose is that every program and human using them will understand it exactly in the same way. To gain this goal, a modeling language like OWL [14], which we are using, is needed.

Since the XML-Syntax is not easy to read, we will use the *Manchester Syntax* [8], that uses natural language, to explain the structure of our SO. Thereby we can describe the set of all audio files (all things that are files and have an audio file format) as Defined Class by writing `File and (hasFileFormat exactly 1 AudioFileFormat)`. Furthermore, the term contains a (normal) *class* `File`. The differences between a Defined Class and a Class is the necessity of inference. While we must explicitly set classes to an *individual*, Defined Classes are inferred by reasoner and we do not have to assign them. There is also an *object property* in the upper term: `hasFileFormat`. Object properties describe the connection between two individuals. Additionally, we will also use *datatype properties* that describes the connection to data values, like string, int or self defined types. An example is `hasSize long`. In this case, the reasoner looks which of all individuals, who are member of the class `File`, has exactly one property `hasFileType` that has the range of another individual as an instance of `AudioFileFormat`, that is also a Defined Class. At last, we have two keywords from the Manchester Syntax: `and` and `exactly`. The keyword `and` is an intersection ($\sqcap$) and `exactly x` ($=$) means that every instance of such a class must have x times a given property. Another important concept of our ontology are *individuals*. These are concrete instances that belongs to concrete classes. While `File` is a class describing an individual belonging to its class, `File123` could be a concrete instance of the class `File` and could optionally have a file name or a size.

2.3 Structure of the SO

Following, we present the schema of our ontology by explaining the main ideas with the help of the Manchester Syntax. At first, we describe the classes and properties belonging to files, and later we want to end up with HSM Systems and the connection between both.

File Concluding from Section 2.1 we modelled the class `File` as the central class (see Fig. 3). There are some datatype properties, among others, that describes

attributes such as path, name, id, size or other important once such as last access or the date of creation. There is also a property `isPreview` that indicates if a file is a preview or not. Furthermore, a collection of files could be added to a `Group`. The connection between `File` and `FileFormat` is more complex. To explain the `FileFormat`, we have to go further to the two classes `MediaType` and `QualityType`. These classes are modeled as closed enumerations. `MediaType` has only five predefined instances based on the MIME Media Types by IANA [9]: `Audio, Application, Image, Video` and `Text`. All the same, we defined the class `QualityType`, that has two instances: `Performance` and `Raw`, considering that a lot of files exists in several characteristics (see Section 2.1). While the `MediaType` could be extracted through several tools, the `QualityType` must be set manually, except of file formats for preview files, because, if a system creates a preview from another file, one could observe this and set the property `isPreview` on true. As an example, `PNG` as instance of `FileFormat` could be created. The two properties `MediaType` and `QualityType` are set to `Image` and `Performance`. A reasoner could infer from this information, that the individual is also a member

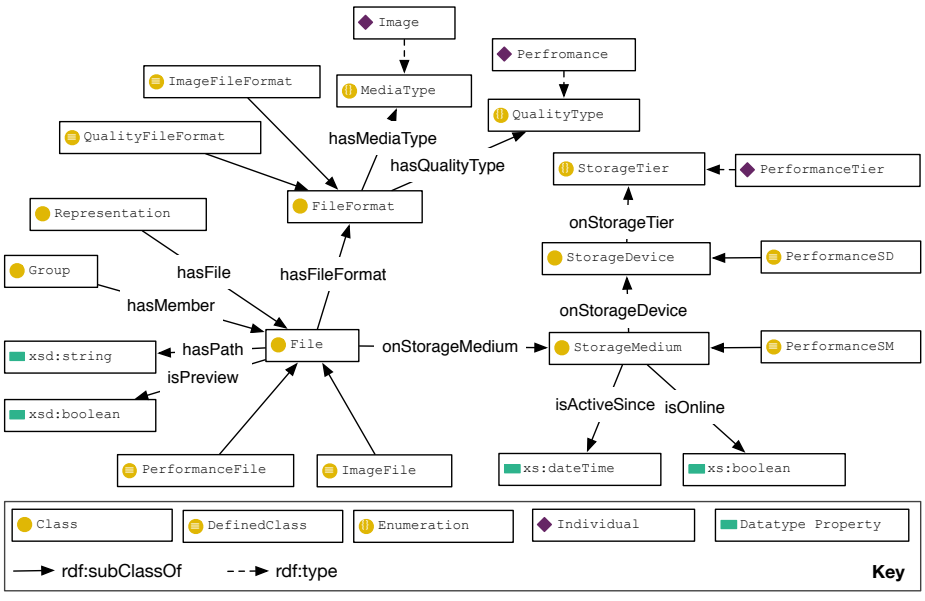

Fig. 3. Partial graph from the SO, that describes raw and image files on the performance tier

Furthermore, an individual, e. g. , `holiday.png` as member of the class `File` would be created. One could use an analyzing tool to get the MIME Media Type from this file and set the property `hasMediaType` to the individual, in this case with the range `PNG`. From this connection, a reasoner could infer that this file also belongs to the classes `ImageFile` and `PerformanceFile`, since the class description is `File and (hasFileFormat exactly 1 ImageFileFormat)` for the Defined Class `ImageFile`. As one could envision, there are also analogous class descriptions for every subclass from `File` that is build from every subclass of `FileFormat`. An exception is the subclass `PreviewFile` as described earlier. This class description is `File and (isPreview true)`.

We also defined the class `Representation` in the SO. This is an abstraction of `Files`, that have at least one similar characteristic. As an example, there could be three different characteristics from one photo as raw photo from a camera, a performance file that was created from the first one and a preview. If an individual of `Representation` would be created and add connections to the three `Files`, one could find these three ones with only one query. As an example, imagine an access on the preview file of a photo, and later the user wants to retrieve the raw file of this photo which is archived. In this case, it is easy to find this photo.

HSM Going on with the classes and properties belonging to the domain of the HSM System, we have made a meaningful basis in Section 2.1 by describing the HSM with the help of EBNF and it is easy to map this into our SO. Therefore, we add a class for every type from the EBNF description, but we omit the class `Storage Vault` in Fig. 1, since it is not necessary if only one is used in the HSM System. According to the classes `MediaType` and `QualityType`, we modelled the class `StorageTier` as an enumeration with the three members `PerformanceTier`, `CapacityTier` and also `ArchiveTier`. With the property `onStorageTier`, an instance of the class `StorageDevice` could be connected to one of the tier, and also with the property `onStorageDevice`, an instance of `StorageMedium` could be placed on a given storage device. Depending on to which kind of storage tier a storage device and on to which storage device a storage medium is connected, a further type could be added to these individuals. Taken the example from Fig. 1, a storage device that is connected to the performance tier would be additionally inferred as a `PerformanceDevice`, because the class description of the defined class `PerformanceDevice` is `StorageDevice and (onStorageTier) PerformanceTier`. We modelled the class description from the defined classes `CapacityDevice` and `ArchiveDevice` and also the different storage mediums, accordingly. Among others, a Storage Device has properties for read/write, access time or throughput.

3 Evaluation

As we focus on the meaning and context of documents we decided to use semantic data technologies instead of a relational database to store extracted information

[13]. This section introduces our test environment, gives an example to compare our approach with the classical one and shows the advantages.

3.1 Test setup and environment

In 2005, we started the K-IMM[2] project [10], which aim is to include the content and context of documents in personal document management. Although the architecture was designed to enable a simple replacement of the used ontology, the prototype supported it in a cumbersome way. Another problem was the support of only one ontology at runtime. Thus, we redesigned the architecture and reimplemented the core module to support multiple ontologies, exchange them at runtime and extended the approach to support small and medium-sized enterprises. We call it KIMM+. In the SENSE[3] project[4], KIMM+ is used as a semantic middleware. Web applications are used to upload documents such as videos and pictures to the SENSE application. These files are analysed by KIMM+ and extracted information is stored in the ontologies, especially the SO. The files are stored on a file system under control of an HSM System. In this infrastructure both, HSM and web applications are able to query the SO without gathering information or access to the file system and the files themselves. Therefore, the HSM System was extended to execute and analyse SPARQL queries. The following section will give two short examples of how to use the SO and highlights the advantages.

3.2 Use cases

In Section 1, we focussed on two aspects: (1) the lack of semantic (e. g., the unknown relations between documents represented by their content) and (2) that classical HSM Systems just use a restricted set of attributes to store files on a specific tier. For the following examples, we assume to have three folders: `music`, `documents` and `pictures`. Each folder contains files related to the topics: `flowers`, `birds` and `research paper`.

(1) In Section 2, we introduced the concept `Group`. We see the SO as a minimal subset of relations necessary to store a file in the best place. Therefore, we need a concept to group files. By keeping it abstract, a group can represent the same topic (e. g. , flowers) or persons in documents. A file can be part of several groups. If a file named `national_fauna.pdf` should be migrated from the PT to the CT, the following query can be used to get the associated groups.

```
SELECT ?group WHERE {
    ?file :hasName "national_fauna.pdf". ?group :hasMember ?file . }
```

The result contains all groups the document is related to. The other documents of these groups can be found by executing for each group:

[2] Knowledge through intelligent media management
[3] Intelligent Storage and Exploration of large Document Sets
[4] `http://www.sense-projekt.de`

```
SELECT ?file WHERE { ?group :hasMember ?file .}.
```

To prevent the HSM System from migrating files that should not be migrated, other attributes have to be considered to get the final result. This is done by the logic of the extended HSM System in SENSE. The example shows that in this way the relations and semantic of documents will be involved in the decision making for migration. We chose the group concept to keep the semantic within the SO as simple as possible, because the HSM does not need to know what a topic is but which files are related by topic. Each concept of grouped entities can be broken down to a SO group.

(2) The second aspect we focussed on is the restricted set of attributes. They are limited to classical attributes to filter files for a policy definition. The folder `pictures` introduced above contains, for example, PNG, TIFF, NEF and JPEG files. If RAW formats can be archived shortly after they where accessed, this is difficult to model with standard policies. First, the attributes to match a RAW format have to be defined. Lets assume TIFF and NEF are RAW formats and need to have a file size greater than 1024 KByte. A policy would like this:

```
(File name matches pattern *.TIFF or *.NEF) and
    (File is larger than 1204KByte) .
```

If the definition of our RAW format changes or a new one needs to be regarded, the policy has to be modified. Using the new approach, the HSM System gets a list of files by executing the following query:

```
SELECT ?file WHERE
    { ?file a RawFile. ?file :hasFileFormat ?f. ?f a :ImageFileFormat. }.
```

In this case, changed requirements can be applied directly to the class definition for `RawFile`. Another benefit is the possibility that files can be excluded in the Defined Class if they match specific criteria. This improves the flexibility of the HSM System.

First tests within the SENSE framework showed another advantage of the Storage Ontology. Listing the size of files, no matter if a group of files or all files are selected, is much faster if the ontology is used. Linux as well as Windows and OS X showed problems, when the HSM filesystem was loaded, containing more than 500.000 files, although they were uniformly distributed in the filesystem. This section showed two small examples how the introduced SO can be used to improve a existing HSM Systems. Currently, we are testing and refactoring the SO within the prototype of the SENSE application under real conditions to verify the improvement. The next sections gives an overview on existing approaches on using semantic technologies to store, search and access files.

4 Related Work

As mentioned in Section 1, we focus on two main problems: the lack of relations regarding to file content and the usage of restricted values such as file size.

Services and platforms like Google Docs, Facebook, Flickr, Youtube and Twitter are used to store personal documents and to share them with others. Even for enterprises this becomes more and more important. Regarding this situation, Semantic Web Technologies can yield to a more flexible way to connect and use these services. Focussing on the requirements of small and medium sized enterprises, we concentrated on the storage of files within a centralized infrastructure. Reasons for such a infrastructure can be required by law. In this context, the Memex Vision published by Vannevar Bush [2] is of great interest. He described a system to handle a large amount of personalized heterogeneous data and defined four requirements, cited among, others by Gemmell et al.[5]: (1) collections and search must replace hierarchy for organization, (2) many visualizations should be supported, (3) annotations are critical to non-text media and must be made easy, and (4) authoring should be via transclusion[5].

We understand (1) and (3) as necessary precondition to fulfill (2), and (4) given by the concept of Semantic Web Technologies. Describing the relations of files using an ontology supports the first requirement. While (semi)-automatic creation from meta data and information retrieval fulfills the third requirement. Having these information in form of an ontology, different visualization can be realized (3). And as shown in Section 2, using Semantic Web Technologies, relations between documents change when rules or schema change (4). A first published approach to store relations between documents and their contents in a ubiquitous way was WinFS [7]. Through an API, the documents and settings folder of Windows Vista was planed to handle files in transactions supported by relational database technologies. The aim was to handle all files in the same way and to get a homogeneous knowledge base on documents on a PC. Although WinFS never was released, parts of the concepts were included in following versions of NTFS on Windows 7 and Windows 8.
An interesting approach to fulfill the first requirement of Vannevar Bush [2] was published by Bloehdorn et al.[1] in 2006. Using WebDAV as an abstraction layer between user and file system enables Bloehdorn et al. to break with the classical hierarchical approach of managing files in folders. To the user, the file system hierarchy is presented, matching results of search queries. The files are sorted and organized in virtual folders depending on their tags. Each Folder represents a tag and a concatenation of multiple tags represents a location. In this way files are tagged by their location with multiple tags. On the one hand, this approach enables an orthogonal search, but on the other hand the subset of tags is limited to the folder names, as they are the tags. That means, that no real search for files ordered by persons is possible.

In [3] Crenze et al. the challenges of information management in enterprise environments are presented. They identified the: (1) amount of data, (2) the quality and performance of extraction tools, and (3) security and authentication of data and data access. Semantic Web Technologies are seen as possibility to replace a classical full-text search by a semantic search on ontologies. According to Crenze et al. a combination of full-text search with semantic-aware filtering

[5] Including documents into each other by using a reference.

and proposal functions is the best combination to retrieve good search results. Because of weak performance, the authors intend to use Apache Lucene[6] to index properties instead of Apache Jena[7].

Regarding the description of files through properties and meta data, PREMIS [11] gives a good system to describe them in a standardized way. Especially, since there also exist RDFS and OWL schemas. Beside a subset of properties to describe data for archiving, PREMIS also covers security and authentication. Although it is designed to achieve a flexible technology-independent way to archive physical objects it is also useful to identify storage related properties for an HSM.

In 2011, we came up with the idea of using a ontology to optimize the placement of files in an HSM System. For the first prototype, we did not use all capabilities of ontologies and used a combination of semantic descriptions and relational rules to generate storage solutions [12]. The great disadvantage of this approach was the modeling of rules that depended directly on information stored in the ontology. Another disadvantage was the minimalistic schema, which was not able to represent storage-related properties. For example, it did not support the description of different file types like raw files that can be stored on cheap but slow memory[8]. In consequence, we focussed on developing the Storage Ontology. The next section gives a conclusion on our results and overview on upcoming work.

5 Conclusion

In Section 2 we introduced a schema to describe files stored on an arbitrary file system controlled by an HSM System. We focused on describing files and their relations among themselves and within an HSM System. The quality of the ontology depends on the used extraction tools to gather information. Using OWL allows a more flexible classification of files such as image files or files which should be moved to the archive. This leads to a more flexible configuration for HSM System policies as the underlying system uses SPARQL queries to get related files. In consequence, the policy engine of existing systems can be used with minimal modifications. The located files are related based on the two mentioned aspect. They are chosen either because of matching attributes or because of relations in content or type. In this way, we introduced a source for decision-making in HSM systems and enabled applications in the front-end to get file specific information as well as knowledge about the placement in the storage hierarchy. We integrated our schema in ontologies developed in the SENSE project. Currently, we evaluate and adapt the Storage Ontology by defined scenarios. The recorded data, like file access-times on the Performance Tier and migration tasks, are compared with the results of the classical approach using the policy engine without semantic. Furthermore, we improve the SENSE framework.

[6] `http://lucene.apache.org/core/`
[7] `http://jena.apache.org/`
[8] Depending on the application domain.

References

[1] Stephan Bloehdorn, Olaf Görlitz, Simon Schenk, et al. "TagFS - Tag Semantics for Hierarchical File Systems". In: *Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06), Graz, Austria, September 6-8, 2006*. Sept. 2006.

[2] Vannevar Bush. *As We May Think*. The Atlantic Monthly. 1945.

[3] Uwe Crenze, Stefan Köhler, Kristian Hermsdorf, et al. "Semantic Descriptions in an Enterprise Search Solution". In: *Reasoning Web*. Edited by Grigoris Antoniou, Uwe Aßmann, Cristina Baroglio, et al. Volume 4636. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pages 334–337.

[4] J Gantz. "The Diverse and Exploding Digital Universe". In: *IDC white paper*. White Paper 2 (2008), pages 1–16.

[5] Jim Gemmell, Gordon Bell, Roger Lueder, et al. "MyLifeBits: fulfilling the Memex vision". In: *Proceedings of the tenth ACM international conference on Multimedia*. MULTIMEDIA '02. Juan-les-Pins, France: ACM, 2002, pages 235–238.

[6] PoINT Software & Systems GmbH. *Automated Storage Tiering: Optimizing the storage infrastructure concerning Cost, Efficiency and Compliance*. http://www.point.de/fileadmin/Documents/White_Paper_Automated_Storage_Tiering_by_PoINT_Storage_Manager.pdf. 2012.

[7] Richard Grimes. *Code Name WinFS*. http://msdn.microsoft.com/de-de/magazine/cc164028%28en-us%29.aspx. Aug. 2004.

[8] Matthew Horridge, Nick Drummond, John Goodwin, et al. "The manchester owl syntax". In: *In Proc. of the 2006 OWL Experiences and Directions Workshop (OWL-ED2006*. 2006.

[9] IANA. *MIME Media Types*. http://www.iana.org/assignments/media-types. 2013.

[10] Annett Mitschick. "Ontologiebasierte Indexierung und Kontextualisierung multimedialer Dokumente für das persönliche Wissensmanagement". PhD thesis. Technischen Universität Dresden, 2009.

[11] PREMIS Editorial Committee. *PREMIS Data Dictionary for Preservation Metadata*. Edited by PREMIS Editorial Committee. www.loc.gov/standards/premis/v2/premis-2-0.pdf. 2008.

[12] Axel Schröder, Ronny Fritzsche, Sandro Schmidt, et al. "A Semantic Extension of a Hierarchical Storage Management System for Small and Medium-sized Enterprises". In: *SDA*. 2011, pages 23–36.

[13] M. Uschold. *Ontologies and Database Schema: What's the Difference*. http://semtech2011.semanticweb.com/programDetails.cfm?ptype=K&optionID=284&pgid=4. 2011.

[14] W3C OWL Working Group. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. http://www.w3.org/TR/owl2-overview/. 2012.

[15] Muzhou Xiong, Hai Jin, and Song Wu. "FDSSS: An Efficient Metadata Management Scheme in Large Scale Data Environment". In: *Grid and Cooperative Computing* 90412010 (2006).