A Metadata-orientated Integrated Approach to Personal File Management

Md Maruf Hasan Chutiporn Anutariya M Zau Ja
School of Information Technology
Shinawatra University
Bangkok, Thailand
{maruf, chutiporn}@siu.ac.th, mzauja@gmail.com

***Abstract*—In recent years, as the computing and storage technology developed and became affordable, different types of systems including mobile devices such as smartphones become part of our daily life. Personal information is now scattered over different systems and devices in different forms and formats. We utilize the file-level abstraction and file metadata to formulate a solution to personal file management. Unlike the widely-used file synchronization and backup where active involvement of a user is essential, our approach relies on a light-weight client program which passively identifies a user activity during any login session in terms of atomic file-event such as file creation, copy, update, and deletion (along with the necessary file metadata). The client synchronizes such information with the server (essentially a metadata repository and optionally, a centralized file-storage). Our approach can easily be extended to support multiple users with emerging computing infrastructure such as the cloud computing in a serious commercial undertaking. In this paper, we will demonstrate our basic idea of metadata-orientated personal file management with a prototype implementation that can handle some simple but crucial personal file management issues (e.g., duplicate detection, version tracking and the like) along with the details of the underlying algorithms. We also elaborate our future plans to address more complex file management scenarios using the same framework.**

Keywords- Personal Information Management (PIM); Resource Description Framework (RDF); File Metadata and File Ontology

I. INTRODUCTION

Personal Information Management is a multidisciplinary research area that addresses issues related to human computer interaction, cognitive science, artificial intelligence, database management and information retrieval. Personal information items include things such as personal files, e-mails, bookmarks, calendar entries, contacts, etc. The ultimate goal of PIM is to reflect the user mental model totally in storing and retrieving the personal information. In this paper we will address personal file management problems that arise from the fact that people often uses more than one computers, and personal files are generally scattered across the systems they use. There is no efficient way to know what files a user owns and where they are located at. Users' files also get easily duplicated across different systems or different folders of the same system. There is a crucial need to organize all user's files at one place in a systematic fashion with support for intuitive search and navigation.

A computer file is a logical entity. Users generally store information in files which are subsequently stored and managed on physical media with the help of an operating system. From the user point of view, files are often organized in a folder hierarchy of the file system. The concept of file is a simple but it is an elegant abstraction. File-related metadata offers rich semantics about file. Such metadata can be efficiently used to help users effortlessly organize and locate all their files at one place taking advantage of the emerging cloud computing infrastructure.

In this paper, we explain a client-server architecture for personal file management that requires minimum user intervention and is able to organize all the files of a user at one place over time. We take advantage of the fact that during any login session all the computing activities of a user (with some minor exceptions) can be aggregated into a set of *atomic* file-events such as file creation (including download or copy), update or deletion. We develop a light-weight client program which authenticates a user at the login and runs in the background. The client passively records all the file activities of a user un-intrusively. At certain interval or during the logout process, the client aggregates and synchronizes all the file-events with the server along with relevant metadata in RDF format. The transfer of file-object may be delayed or may be triggered by the server using pull technology. The server is essentially a metadata repository but it can also be configured as a centralized file-storage for any user (identified by a unique *userID*). The server program is capable of handling SPARQL query [14] and RDFS++ reasoning using RDF Schema [17] and a File Ontology (a modified version of Nepomuk File Ontology [3] is used with our prototype). The server is capable of aggregating and organizing all user files dynamically by identifying duplications and associations among all files from every system a user may use (as long as a client program is available and installed on the system).

The prototype implementation and algorithms explained in this paper are mostly based on a single user scenario. However, the metadata-oriented integrated personal file management framework explained here can easily handle multiple users. If we further consider the fact that users generally do not create very many unique files on a daily basis but do accumulate large numbers of files over time, there is a significant overlap in file ownership among users. If our approach is extended to commercial multi-user level implementation, the server can easily take advantage of multi-user file sharing and dramatically reduce the storage required to support millions of users.

In the following section of this paper, we review some relevant work. We subsequently elaborate the file metadata and file ontology used in our prototype in Section **III**; and the prototype development itself is explained in Section **IV** along with the algorithms we implemented. Section **V** analyzes some advanced scenarios along with their potential applications in personal file management. Finally we conclude the paper in Section **VI**.

II. Related work

In this section, we review only two of the relevant research projects which are historically significant. Both **MyLifeBits** [4] and **ROMA** [5] address the challenges in personal information management with ambitious goals as elaborated below. Our approach however is based on the simple hypothesis (i.e., file level abstraction and file metadata) to organize user's files all at one place identifying duplicates and relationships. We try to integrate up to date computing technologies to formulate solutions for PIM keeping users' needs, preference and lifestyle in mind. We deliberately omitted several popular projects and products in file-sharing, backup and synchronization (including DropBox) that require active user intervention and create more problems than solutions as personal files get duplicated and scattered.

A. *MyLifeBits*

This is a system originally developed using SQL server database at its heart for storing personal information from captured, scanned and encoded material such as articles, books, photo and video as well as digital media such as office documents, emails and digital photos. Later it evolved to store all the information that can be captured in a person's life time including heartbeat and blood pressure.

After storing all the content and metadata from various types of application such as contacts, documents, email, events, photos, songs, and video in the SQL database, it tries to organize the items, all captured entities exposed to the user, in a single huge folder rather than categorizing in many folders and sub-folders.

The solution is that classifying all the items based on the content which was divided in four ways using two orthogonal attributes, time and lives. The upper layer is classified based on the professional life and person life. Below that it divided into current and archive. Past personal life e.g., past college trip is stored in the archive but for professional life is ongoing nature and it will be stored in the current sub-folder.

Although MyLifeBits supports full text search over the metadata structured in the database schema, it is not enough because many items required enough attributes to be found. Therefore, they also use hierarchical classification over the "document type" which contains several hundred unique attributes such as article, bill, will, business card, report card, greeting card, and birth certificate. When a collection of items becomes so large, a user may not remember much about the content of the items. As a solution, killer application screen saver is used in showing the photo and video clips previews.

B. *ROMA Personal Metadata Service*

Roma manages user's files located in different devices or systems on a metadata level. In Roma, the metadata server is a centralized server and stores information about a file or file attribute or metadata such as name, location, time-stamp and keywords. However, the data file is stored in distributed environment including file servers, PDA (Personal digital assistance), web servers, cell phones and any electronics devices that have storage location. Separating the metadata from the data repositories make highly available to the metadata server.

In ROMA, the application is responsible for presenting the metadata explicitly and automatically to the user to make the decision on the file version of the files. The data location is specified by the metadata URI and a file version number keeps track of every changes made to a file instance. Moreover, when file changes occur, metadata server needs to be informed by Roma synchronization agents so that the file instances remain consistent.

When updating the files, the old version from different clients are up-to-date to the latest one by synchronizing. For the duplicate files, Roma makes the unique file identifier that is common to all instances of all related copies. Unlike ROMA, where emphasis is given to track changes, we utilize file metadata in addressing overall PIM issues using a client-server approach as elaborated in the rest of this paper.

III. File Metadata and Ontology

A. *Basic and Application-specific Metadata Extraction*

As we focus on file level abstraction in PIM, file metadata is mainly used in our system. Metadata serves as resource discovery as good cataloging does by describing resources [7]. In this system, metadata is a key and raw material for the processing. Basically file name, file size, file hash, file type and file path are included. We extract both the basic metadata, and the application-specific metadata (e.g., image height, image width, etc. for an image) as shown in table 1 to 4 . Application -specific metadata are used in effective retrieval of files through association or other relationship as defined in the file ontology.

TABLE I. List of Basic File System Metadata

Metadata item	**Description**
fileName	Name of the file
fileHash	Hash value of the file
fileType	Type of the file
filePath	Path of the file
hostName	Name of computer or device
userID	The ID of user who owns the file
copyCount	Number of file copies
fileVer	Modified version number of each file
Deleted	To flag deleted file
fileID	Unique file ID
fileCreated	Created date and time of the file
fileLastModified	Date and time of last modified

fileDeleted	Date and time when the file was deleted

TABLE II. LIST OF APPLICATION-SPECIFIC METADATA FOR IMAGE

Metadata item	Description
tiff:Image height	Height of the image in pixels
tiff:Image width	Width of the image in pixels
tiff:Date time	Time stamps of the image
tiff:Xresolution	Horizontal resolution in pixels per unit
tiff:Yresolution	Vertical resolution in pixels per unit
tiff:Make	Manufacturer of recording equipment
exiff:Focal Length	Focal length of Lens, in millimeters
exiff:DateTimeOriginal	Date and time when original image was generated.
tiff:Model	Camera model taken image.
exiff:Flash	Strobe light (flash) source data.
exif:DateTimeDigitized	Date and time when original image was stored as digital data

TABLE III. LIST OF OLE METADATA FOR MSOFFICE FILES

Metadata item	Description
Author	Original author of the document
Creation date	Created date of the document
Last author	Author last modified and saved
Last saved date	Date saved by last author

TABLE IV. LIST OF MUSIC METADATA

Metadata item	Description
xmpDM:ReleaseDate	Date and time release the song
xmpDM:Album	Album of the song
dc:Author	Author of the song
xmpDM:Artist	Artist of the song
dc:title	Title of the song
xmpDM:Composer	Composer of the song

B. *File Hash calculation*

For the duplicate detection, encryption hash value is widely used in computer forensics. MD5 [12] 128-bit hash value is enough for the detecting full duplicate. Each file has a hash value based on its file content. Therefore, it is used together with other file system metadata to detect file duplicate.

C. *Metadata Encoding and Repository*

File metadata is encoded in RDF [1] and is stored at the server to allow semantic metadata processing. RDF metadata repository [2] is used to handle the metadata request and response generated from the RDF query language (SPARQL) [14] over the HTTP. All the extracted metadata are extended to the vocabulary of Nepomuk file ontology which is already standard as part of the Nepomuk Social Semantic Desktop [13].

D. *File Storage*

When file objects from the client computers are stored at the server, dynamic file classification is possible with the help of file metadata. File organization using folder and sub-folder structure at the server side is therefore not necessary. Server stores all files of a user in a flat list of file objects with unique fileID. In this way, both the system and the user can find all of a user's files at one place regardless of how many systems and devices they use.

IV. PROTOTYPE DEVELOPMENT

To show the proof of concept of this work we have implemented mainly core part of the metadata processing.

A. *Integrated PIM Architecture*

Fig.1 shows the proposed architecture which is based on a client-server model.

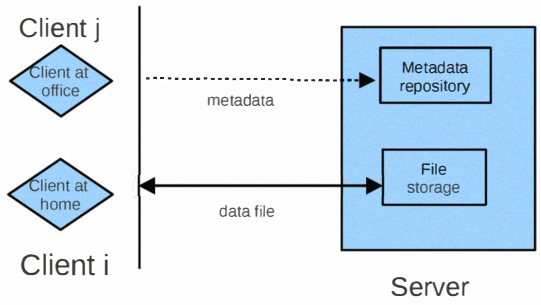

Figure 1: Integrated PIM architecture overview

The client is a light weight application running in the background when a system starts up. It passively tracks and records all the file activities including file creation, deletion and modification. Basic file system metadata and application-specific one are extracted and sent to the server for updating the RDF metadata repository[15]. Algorithm 1-3 explain how to deal with the file creation, file modification and file deletion activities, respectively.

Algorithm 1. File Creation

Definition: Copy, download, and creation of a file from an application trigger the file-creation event on the file system. CopyCount metadata applies to all the duplicate files in all computers of a user. It creates a new CopyCount and file version if there is no duplicate file in the server.
Input: fileChange, extracted file metadata- fileHash, fileName, fileType, filePath, hostName, userID
Output: fileID, CopyCount,maxfileID, fileVer

```
[1]  copyCount ← 0
[2]  fileID ← 0
[3]  fileVer ← 0
[4]  if fileChange = "Create" then //in case of file create
[5]    Search the "files_k"for the file duplicate in metadata record of metadata
     store where fileHash is equal. //files_k = set of query returned result
[6]    if files_k is not null then
[7]      for each k in files_k do   // for each duplicate file
[8]        hostName' ← files_k [hostName]
[9]        userID' ← files_k [userID]
[10]       filePath' ← files_k [filePath]
```

```
[11]        fileName' ← files_k [fileName]
[12]        copyCount ← files_k [copyCount] + 1      //update copy count
[13]        fileID ← files_k [fileID]      //fileID as returned fileID
[14]        if fileName = fileName' and hostName = hostName' and
            userID= userID' then  //duplicate file with the same file name in
            different path of same host
[15]            Update the metadata record using fileID for the changed
                copyCount, different filePath and file time-stamps
[16]        else if filePath=filePath' and hostName=hostName' and
                userID = userID' then   //duplicate file with different file
                name in same path of same host
[17]            Update the metadata record using fileID for the changed
                copyCount, different fileName and file time-stamps
[18]        else if fileName=fileName' and userID=userID'
          //duplicate file with the same file name in different hosts
[19]            Update the metadata record using fileID for the changed
                copyCount, different filePath, hostName and file time-stamps
[20]        else if hostName=hostName' and userID=userID
          //duplicate file with different file name in different path of same host
[21]            Update the metadata record using fileID for the changed
                copyCount,different fileName,filePath and file time-stamps
[22]        else //duplicate file with different file name in different path of
                different host
[23]            Update the metadata record using fileID for the changed
                copyCount, different fileName, filePath, hostName and file
                time-stamps
[24]        end if
[25]    end for
[26]  else          //the file is not duplicate
[27]        Transfer the file to the file storage.
[28]        Search the maxfileID in the metadata-record of metadata
            store.
[29]        fileID ← maxfileID + 1
[30]        copyCount ← 1
[31]        fileVer ← 1
[32]        Update the metadata record with fileID, copyCount, fileVer
            and all other extracted metadata.
[33]    end if
[34] end if
```

SPARQL Query:

```
PREFIX nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#>
PREFIX ex-nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/ex-
nfo#>
SELECT DISTINCT ?fileID
  WHERE {
      <http://www.filesemantic.org/v1>
      ex-nfo:fileID ?fileID;
      nfo:fileHash  ?fileHash.
      FILTER  (
          ?fileHash = '%s'
           ) } LIMIT 1
```

Result : fileID – unique id of a file returned from repository if it is a duplicate with the same hash

Algorithm 2. File Update

Definition: Modify the file and save the file on the file system. If the file is duplicate, increase the file version number; the CopyCount is not changed.
Input: fileChange,extracted file metadata: fileHash, fileName, fileType, filePath, hostName, userID
Output: fileID, fileVer

```
[1]     copyCount ← 0
[2]     fileID ← 0
[3]     if fileChange = "Update" then
[4]         Search the "file" for the duplicate in metadata record
            of metadata store where fileName, fileType, filePath,
            hostName and userID are equal.
            //check the duplicate file with the same name in same path and host
[5]         if file is not null then
[6]             fileVer ← file[fileVer] + 1
[7]             fileID ← file[fileID]     //assign the fileID as the returned id
[8]             Transfer the file to the file storage
[9]             Update the metadata record as the returned fileID with
                changed fileVer and all other extracted metadata
[10]        end if
[11]    end if
```

SPARQL Query:

```
PREFIX nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#>
PREFIX ex-nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/ex-
nfo#>
SELECT DISTINCT ?fileID
  WHERE {
          <http://www.filesemantic.org/v1>
           ex-nfo:fileID ?fileID;
           nfo:fileName ?fileName;
           nfo:fileHash  ?fileHash;
           ex-nfo:fileType ?fileType;
           ex-nfo:filePath ?filePath;
           ex-nfo:hostName ?hostName;
           ex-nfo:userID ?userID.
    FILTER (
       ?fileHash = '%s'  &&
       ?fileName = '%s' &&
       ?fileType = '%s'  &&
       ?filePath = '%s'   &&
       ?hostName = '%s'  &&
       ?userID = '%s'
         ) } LIMIT 1
```

Result : fileID – unique id of a file returned from repository if the file is already existed in the server with the same file name,file type,file path and computer

Algorithm 3. File Deletion

Definition: Deletion of a file: if the file is a duplicate in one of computers, it will reduce the CopyCount value. When the CopyCount value becomes zero, mark the Deleted flag as 1.
Input: fileChange, extracted file metadata: fileHash, fileName, fileType, filePath, hostName, userID
Output: fileID, copyCount, Deleted

```
[1]     copyCount ← 0
[2]     fileID ← 0
[3]     if fileChange = "Delete" then
[4]         Search the "file" for the file duplicate in metadata record
            of metadata store where fileName, fileType, filePath,
            hostName, userID are equal.
            //check the duplicate file with the same name in same path and host
[5]         if file is not null then
[6]             copyCount ← file[copyCount] - 1
[7]             fileID ← file[fileID]
[8]             if copyCount = 0 then
[9]                 Deleted ← 1  //assign deleted flag 1 in case of copy count 0
[10]            end if
[11]            Update the metadata record as the returned fileID with
                changed copyCount, Deleted.
[12]        end if
[13]    end if
```

B. File Retrieval

Basically, browse and navigation are the common mechanisms currently supported by all modern operating systems. With browsing, user can get near the vicinity of the targeted files. However, browsing becomes inefficient when there are thousands of files in a user's disposal; and it becomes cumbersome to locate the file. Therefore, search and navigation are preferable and effective in locating files. We are currently considering how to effectively use file metadata to organize

personal file dynamically and to facilitate effective search, browsing and visualization.

C. *Limitations*

Although our PIM architecture meets the requirement to solve the critical problem of file duplication in many different computers with extended features to be discussed in section V, there remains some issues to be addressed: when the running client is in offline mode to the server,our approach cannot update to the metadata repository and cannot transfer the file to the file storage. In future work, we will extend the design issues in disconnected client to the metadata server by logging the metadata of all the files which was created, modified and deleted in offline period and keeping the list of data files to be transferred in time of the client becomes connected to the server.

V. Extended Features and Future Work

Based on the approach we described earlier, the application can be used and improved by adding more features as described in the following scenarios.

A. Scenario 1 (duplicate file from the same computer)

A user may have duplicate files on a single computer at the same location (the same file path but different names) or different location (with the same or different name). If the file already exists in the file storage, files become duplicated in the server. By detecting the duplicate file, the server's storage space can be reduced up to a certain limit. In case a user edits the duplicate file, the file version number will increase and keep the file for that version in the file storage so that it can support the user to retrieve the files with the latest version easily.

SPARQL Query:

```
PREFIX nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#>
PREFIX ex-nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/ex-nfo#>
SELECT DISTINCT ?fileID
  WHERE {
          <http://www.filesemantic.org/v1>
            ex-nfo:fileID ?fileID;
            nfo:fileHash  ?fileHash;
            ex-nfo:hostName ?hostName;
            ex-nfo:userID ?userID.
       FILTER  (
          ?fileHash = '%s'  &&
          ?hostName = '%s'  &&
          ?userID = '%s'
         ) }
```

Result: fileID – returned ids of all the files that are duplicate in the same computer

B. *Scenario 2 (duplicate file from different computers)*

A user may create duplicate files in different computers where the file is already stored in the server. Sometimes a user creates duplicate files intentionally in many different devices or locations for backup and ease of retrieving the file. The server detects the duplicate file and will not store the file but its metadata record will be kept in the metadata repository.

SPARQL Query:

```
PREFIX nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#>
PREFIX ex-nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/ex-nfo#>
SELECT DISTINCT ?fileID
  WHERE {
      <http://www.filesemantic.org/v1>
         ex-nfo:fileID ?fileID;
         nfo:fileHash  ?fileHash;
         ex-nfo:userID ?userID.

  FILTER  (
       ?fileHash = '%s'  &&
       ?userID = '%s'
       ) }
```

Result: fileID – returned ids of all the files that are duplicate in different computers

C. *Scenario 3 (file classification based on file type)*

Sometimes a user may want to view the files not only one file type but also many file types under the same category. Basically a user can view all the document files or all the image files or all the video files. It can be classified by all the files types to their relevant organization. In document files, there can be many file types e.g., txt, rdt, doc and docx.

After sorting out all the possible file extensions under the same category such as photos, documents, presentations, videos and sounds and then query to the metadata repository. Even in the document types, it can be classified such as bill, invoice, notes, certificate and so on.

SPARQL Query:

```
PREFIX nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#>
PREFIX ex-nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/ex-nfo#>
SELECT DISTINCT ?fileID
  WHERE {
  <http://www.filesemantic.org/v1>
        ex-nfo:fileID ?fileID;
        ex-nfo:fileType ?fileType.
    FILTER  (
      ?fileType = 'jpg'  &&
      ?fileType = 'png'  &&
      ?fileType = 'jpeg'  &&
      ?fileType = 'bmp' &&
      ?fileType = 'tiff' &&
      ?fileType = 'gif'
       ) }
```

Result: fileID – returned ids of all the files that are classified as image type based on the metadata.

D. *Scenario 4 (temporal view)*

A user frequently searches and retrieves the files based on the temporal attributes such as file last modified time, file created time and file last access time in current operating system. It is quite useful for the user in getting the files if he remembers the exact date and time. However, sometimes they created or modified the files long times ago and can only remember the estimated period. It is more efficient to retrieve files by giving an approximate date/time such as yesterday, the day after yesterday, two weeks ago, five months ago and two years ago. Not only that it should also support the users in retrieving files according to the name of the day and month such as Sunday, Monday, January and February, etc.

E. *Scenario 5 (file in the same context)*

Sometimes a user may create and modify many files at the same time or approximately the same time. We may assume that if two or more files are updated or modified during the same login session, they are probably in the same project or context. All the files in the same context have *indirect* relationship. In such a case, all these files have the same metadata value "fileSessionID".

F. *Scenario 6 (task oriented hierarchical file system)*

Currently most client file systems use hierarchical structure. Our study also shows that users often prefer a task-oriented classification to manage and organize the files [18]. In hierarchy, the explicit relationship between the logical structured folders can be captured. File path can explicitly have the semantics of the file classification task [6] e.g., in "C:\Software development\ project1\budget\" file path, there can be many projects under the folder of Software development and each project has each own budget folder and related files. A user can easily infer the files by differentiating the folder paths. Such kind of advantage should be merged into our system.

G. *Scenario 7 (alert message for updating outdated file)*

When a user has duplicate files in many different devices, they may update the file from one computer but the duplicate files from other computers will remain in the old version. Therefore, it should make an alert to a user when he opens the duplicate file with an outdated version and user should have choices to download the updated files from the server.

H. *Scenario 8 (make reference to the original file)*

There are some minor flaws in the file creation algorithm. When a user opens a file, modify it and save it as a new file in another location with the new file name, the current algorithm regards it as a new file and will not carry the "fileVer" from the old file. As these two files are related, the algorithm should make reference to the original one.

This scenario has to be improved so that a user can easily see all the *direct* related files from all computers. As one of the solutions, some application-specific metadata carry the same value to the derived files. For example some files which have XMP metadata [16], "xmpMM:DerivedFrom" metadata can be used in this case. However, this metadata cannot be seen in all the file types. As far as we have tested, Microsoft OLE [19] metadata also has "Author" and "Created Date" which remain the same in all derived files. Moreover, these two metadata can be seen in different kinds of metadata encoding schema. Therefore, if a file has the same original author and creation date-time metadata, it can be said that these two files have the same reference to each oth

VI. CONCLUSIONS

Our proposed approach is not merely the file synchronizer or backup, it is trying to improve the current solutions for personal file management problems based on the metadata. Although we use the term "Personal" here, now we have not covered all the personal information yet. As a future work, we will try to cover all kinds of personal information in PI (Personal Information) space. Last but not least, utilizing the metadata kept in server part and SPARQL query; we will implement and improve the scenarios described in Section V.

REFERENCE

[1] W3C(Feb.10,2004), Resource Description Framework(RDF): Concepts and Abstract Syntax, Retrieved from (visited Aug,2011) http://www.w3c.org/TR/rdf-concepts/

[2] A. Ran, and R. Lencevicius, "Natural Language Query System for RDF repositories," Proceedings of the Seventh International Symposium on Natural Language Processing, 2007.

[3] Nepomuk file ontology (2007). Retrieved from (visited Aug 2011): http://www.semanticdesktop.org/ontologies/nfo/

[4] J. Gemmell, G. Bell, and R. Lueder, "MylifeBits, a personal database for everything," CACM publication, Microsoft Bay Research Center, Sanfransico, CA, January 2006.

[5] E. Swierk, E. kiciman, V. Laviano, and M. Baker, "Roma Personal Metadata Service," The third IEEE workshop on mobile computing system and application, December 2000.

[6] Indratmo, and J. Vassileva, "A review of organization structure of personal information management," Journal of Digital Information, vol.9, no.1, 2008.

[7] National Information Standards Organization Press, " Understanding Metadata," 2004. Retrived from: http://www.niso.org

[8] W. Jones, and J. Teevan, "Personal Information Management," Seattle:University of Washington Press, 2007.

[9] R. Bordman, "Improving Tool Support for Personal Information Management," Department of Electrical and Electronics Engineering, Imperial College London, University of London Press, July 2004.

[10] S. Faubel and C. Kuschel, "Towards Semantic File System Interface," 7th International Semantic Web Conference, 2008.

[11] W. Jones, and H. Bruce, "A report on the NSF-Sponsored Workshop on Personal Information Management," Seattle, WA,2005.

[12] E. Thompson, "MD5 Collisions and the impact of computer forensics," Digital Investigation, 2005.

[13] T. Groza, S. Handschuh, K. Moeller, G. Grimnes, L. Sauerman, E. Minack, M. Jazayeri, Ce´dric Mesnage,G. Reif, and Ro´sa Gudjo´nsdo´ ttir, "The Nepomuk Project, On the way to Social Semantic Desktop,".

[14] W3C(Jan.15,2008), SPARQL Query Language for RDF: Retrieved from (visited Aug,2011) http://www.w3.org/TR/rdf-sparql-query/

[15] S. Harris, "SPARQL query processing with conventional relational database system," International Workshop on Scalable Semantic Web Knowledge Base Systems, 2005.

[16] Adobe System Incoporated, "XMP Specification," September 2005. Retrieved from (visited Aug,2011) http://partners.adobe.com

[17] W3C(Feb.10,2004), RDF Vocabulary Description Language 1.0:RDF Schema, Retrieved from (visited Aug,2011) http://www.w3.org/TR/rdf-concepts/

[18] D. Barreau, and B. A. Nardi, "Finding and Reminding," SIGCHI Bulletin, vol. 27, no. 3, pp.39-43, 1995.

[19] J. Migletz, S. Garfinkel, and K. Squire, "Automated Metadata Extraction," Naval Post Grauduate School Press, Monterey, California, June 2008.