Automatic File Indexing Framework An Effective Approach to Resolve Dangling File Pointers

Yasas Diniesha Jayaweera
Department of Information Technology
Sri Lanka Institute of Information Technology

ABSTRACT
Today managing files in a server system has the same magnitude as managing the World Wide Web due to the dynamic nature of the file system. Even searching for files over the file system is time consuming because finding a file on hard disk is a long-running task. Every file on the disk has to be read with dangling pointers to files which no longer exist because they have been changed, moved or deleted. This makes the user frustrated. The Automatic file indexing framework facilitates users to resolve file names and locate documents stored in file repositories. The main design objective of the framework is to maintain sub-indexes at the folder level that have the full knowledge of the revisions that are made at the folder level automatically.
This research proposes a framework that manages the creation and maintenance of the file index, with the use of Resources Description Framework (RDF) and retrieval using semantic query languages i.e. SPARQL. The sub-indexes are maintained hierarchically starting from the leaf node to the root node recursively. The proposed framework will monitor the file system continuously and update individual folder descriptors (sub-indexes) stored on each node as the file system changes making the cached indexes resilient to any file changes. The framework is resilient of file or folder name changes. Further, the study explores avenues to build an offline semantic index that can be used by the clients to perform distribute file search without performing the search on the server itself. This is viable since the framework uses semantic languages to describe and build file descriptors that can easily integrate semantic indexing and hence this makes the index readily available for the Web.

General Terms
File Indexing, File Retrieval, Knowledge Management, Semantic Web Technologies.

Keywords
File Indexing, Document retrieval, Semantic Web, RDF (Resource Description Framework), SPARQL.

1. INTRODUCTION
Today, there is an enormous number of files stored on Web servers and even on Personal Computers (PC). The PC has now become a Web of files. The growth of the file system is exponential. Yet, it is a herculean task to locate the stored files manually in a timely manner. To reduce the time spent on manual finding there are search tools which facilitate a smart finding of files to users. In the context of an application to locate files it uses a file index.
There are many index creation tools to avoid user frustration. Manually retrieving a file is a tedious task unless one can remember the location of the files stored. Due to the rapid increase in files in the system it is the job of the file indexing tools to provide an interface to applications which help locating and retrieving the files the user wants.
Current tools in the market improve searching via manual or automatic indexing which facilitates fast retrieval of the files but dangling file pointers still exist making it difficult to locate the file when the location or name changes. Most windows and Web based applications index recently used file entries as a quick reference for the users. But when a file is moved, changed or deleted the entries in the index remain unchanged leaving the entries dangling and pointing to null references leading to user frustration. This is due to the absence of a central index or because of an obsolete index. Either type will be of no use to the user since the user has no clue of retrieving the file if the file has been moved or the path has been changed.
Building an index from scratch takes a lot of time depending on the size of the file system. After the index is built the index should be up to date by incorporating file system changes. Unless it is updated in a timely manner the index becomes obsolete. There are many indexing tools existing in a system but they do not survive when the file is moved, changed or deleted. Tools like Windows 7 search and index, facilitate users to find what is necessary via its search interface. The search result can be indexed for later retrieval by storing the query used to search the files but there is no metadata involved in maintaining the file system changes. Therefore, the index is unaware of the changes to the file system. As such, the problem persists when the file system changes. The stored query retrieves the file system with new changes but the application may refer to the old index references making the changed entries to dangle around. This can happen due to one of the following reasons: file has been deleted, file name has been changed, file has been moved or file path has been changed. If the index system does not maintain metadata about the file changes it will not be able to identify what has caused the system to fail while retrieving the file. For instance if a file name has been changed then the application uses the previously indexed, old symbolic file name which does not exist in the current file system. At a glance one can say the indexes help applications to locate what the user really wants but without maintaining metadata about file changes the index becomes obsolete where it cannot locate files once the file system has changed.
The primary objective of the proposed framework is to trace path and file changes in a metadata file stored locally in each folder in the folder hierarchy. It helps applications to locate a file if the file exists. So that the user does not have to remember file name or file location changes making the file retrieval system intelligent.
The proposed framework will further incorporate file system changes automatically. That is; in the existing environment when files are copied from one location to another the index should be rebuilt manually to make the index up to date. But

in the proposed approach since each leaf node maintains a sub-index only the root node index has to be adjusted accordingly. The framework proposed will maintain each sub-index by adhering to the local changes made to the file system leaving the index up to date.

This work is organized as follows. Section 2 introduces related work and background of the research problem. Section 3 reviews major components of the proposed Automatic File Indexing Framework. Section 4 explains the detail implementation of the framework. This section also gives an insight into the working prototype followed by Section 6 where results are analyzed. Section 6 summarizes the framework with a detailed discussion and is followed by future work.

2. RELATED WORK

There exist many approaches to deal with file indexing. Here I review a few approaches available in the market.

Google Desktop Search [1] is a local search engine that facilitates file search and background indexing. Its primary objective is to locate the file fast. No attention is paid to maintain symbolic name changes.

Operating systems like Windows Vista and Windows 7 [2] [3] facilitate search as an inbuilt feature in the OS. The search result can be saved for later reference. The stored file saves the query which was used to search the files in XML format. When the user runs the stored XML file the operating system re-runs the query on the Windows search subsystem. As a result there is no actual index; rather a directory search. Similar to Google Desktop search the tool will not be able to track the file system changes separately.

There are systems that remember files a user has visited and allows a user a quick access to the files recently visited. A related approach under this is the "Stuff I've Seen" system by Dumais [4], which simply remembers all entities including files, Web pages, emails, contacts, etc. that a user comes into contact on a computer. By revising this history, which is like a super-charged Web browsing history, a user is able to find items guided mainly by his temporal recollection of the desired item. But like in the previous methods this lacks semantics about changes in the file system.

A less related but an effective system is Essence: [5] a resources discovery system on file indexing that exploits file semantics. It extracts keywords that summarize files and generates a compact index. It automatically generates the index discovering different types of file resources as its main design objective but less attention is paid to maintaining the index overtime by taking file system changes to account.

Most of the file retrieval systems have incorporated file event logging for audit purposes [4][6][7]. But none of the systems uses the event log to resolve file name changes in retrieving files. In these systems the user can separately view the log for audit purposes mainly to retrieve statistics. The Automatic File Indexing Framework uses automatic event logging and tracking to facilitate file retrieval irrespective of the changes to the file system.

The proposed framework; Automatic file indexing framework has roots to file indexing and symbolic name maintenance framework [8]. The framework allows pluggable index handlers and also provides a mechanism to keep track of symbolic name association for every file in the system. It provides session based and transient shadow table of symbolic names previously used by the files. The proposed framework takes a different approach in taking file symbolic names beyond the life span of a file. It uses Resource Description Framework (RDF) to describe and maintain metadata about each directory at the folder level allowing changes to be manageable. To make the file retrieval process feasible a hash table is also maintained. Further, the framework can dynamically adjust physical path of the files making it resilient to folder name changes. So the proposed framework enhances functionality with the handling of dangling pointers and resolves directory paths.

3. AUTOMATIC FILE INDEXING FRAMEWORK

The proposed framework functions in two ways. Firstly, the framework maintains sub-indexes at the folder level. Secondly, when applications request files the framework resolves the path and file names to locate the files.

The key design goal of the proposed framework is to handle dangling file pointers and resolve any path changes. At any given time applications are able to access the files irrespective of any naming changes to files or directories in the physical path. The framework decouples the index maintenance process from the file system and the update mechanism provides a flexible approach to the problem of dangling pointers with the problem of allowing applications to refer the files if the file exists irrespective of changes to the file name or directories. Further, the framework is implemented in the user address space without modifying the rest of the system or can be moved to a new system where it increases the flexibility of the framework.

Automatic file indexing framework is designed to maintain folder descriptors at each folder level. At the folder level each folder is able to maintain the metadata file to track file and folder changes. The framework uses Resource Description Framework (RDF) [9] to maintain the folder descriptors. RDF can be used to store metadata of any Web of data. The proposed framework continuously watches the file system changes. When a change occurs the framework receives a notification from the file system notifying the change and the physical location of the place that took the change. Based on the notified change the RDF file (folder descriptor) is maintained to reflect the change. In addition to the indexing, a hash index table is also maintained as a fast lookup to make file retrieval process a computationally viable approach. The framework consists of six (6) main components as shown in figure 1. The following section describes the functionality of each component.

3.1 File System Monitor

The File System Monitor component provides the main interface that interacts with the row file system. In the proposed framework it is required to register a specific root of the file system before receiving notifications of the file system changes. This gives flexibility to enable or disable the automatic indexing of the target file system. Once a root is registered with the File System Monitor, it continuously receives changes in the target file system. Changes like creating files, deleting files, changing file name and changing directory names generate notifications from the target file system. The component intercepts the notifications and queues them for further processing.

3.2 Localizer

The Localizer component plays a main role in the automatic file indexing framework. Its process is of two stages. At the first stage the Localizer, segments and identifies each notification type that is: create, update or delete and the location of the place where the change took place. This information helps the framework to categorize each notification and enables it to use a specific handler to deal with each event. The following actions are taken by event handlers based on the type of the event.

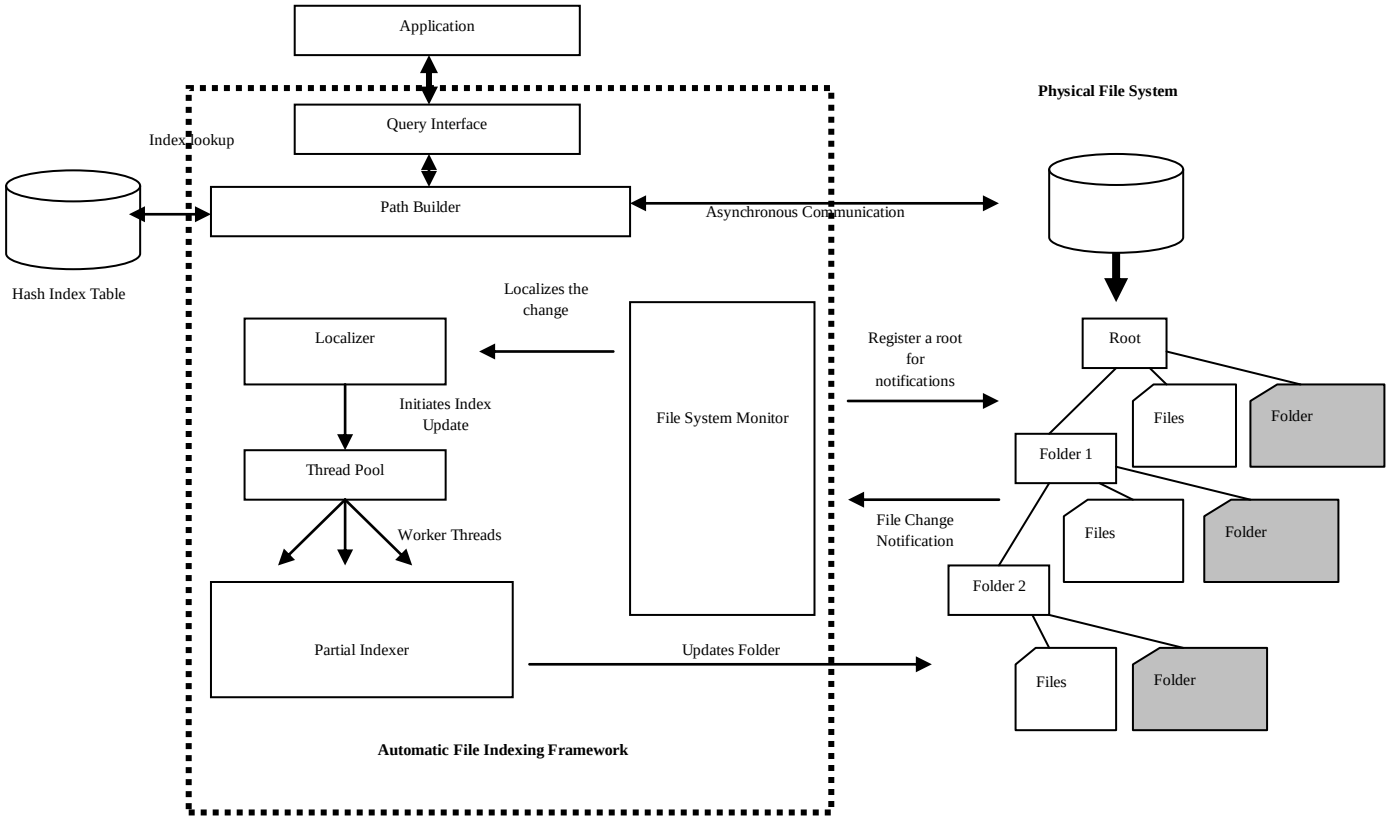

Figure 1: Automatic file indexing framework

- Create a new file: - Handler creates a new entity in the sub-index.
- Delete a file: - Handler deletes the entity in the sub-index.
- Update file name**:** - Handler updates the path by modifying the entity in the sub-index.
- Update location: - Handler updates the path iteratively by modifying the path from root to the leaf.

In the second stage the component retrieves the location where the event was triggered. The leaf node of the path of the event triggered has the folder descriptor which is the sub-index file. The sub-index maintains entities in a RDF file where each entity in the directory is mapped in to the RDF file as a resource. The folder descriptor has the revisions done to file names, deletions and new entries. The component assigns the handler and the path to an available worker thread in the thread pool to run the partial indexer that builds the sub-index.

3.3 Partial Indexer

The Partial Indexer component maintains sub-indexes at the folder level. Each directory contains a folder descriptor; a metadata file that describes the contents of the directory including revisions done. The metadata file (folder descriptor) is a RDF file which maps the directory contents to a RDF file. The "index.rdf" is added to each folder as a hidden file to avoid any accidental deletions. In case the "index.rdf" file is deleted the file is reconstructed with the changes via the hash index table.

Resource Description Framework or shortly RDF [9] is used to describe resources. Semantic Web applications, and in relatively popular applications of RDF like RSS and FOAF (Friend of a Friend), resources tend to be represented by URIs that intentionally denote, and can be used to access, actual data on the World Wide Web. But RDF, in general, is not limited to the description of Internet-based resources.

Each entity in the directory is mapped into a resource. Each resource has a set of names as properties. Following is a snapshot of "code" directory and the sub-index file. Figure 2 represents the directory contents.

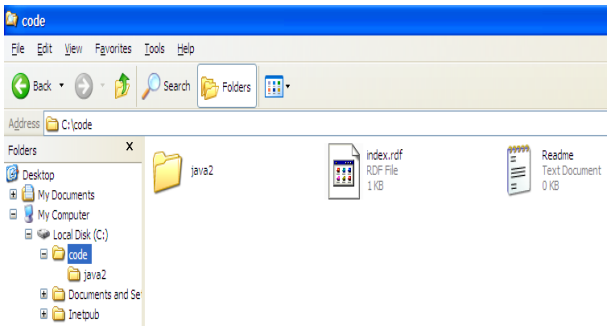

Figure 2: View of the code directory

Figure 3 represents the content of the "index.rdf" the metadata files' structure.

File System Monitor, Localizer and Partial Indexer facilitate automatic indexing of the file system. The next section covers the main components used in the process of file name and path resolution to cater to application file retrieval requests.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:Folder="http://www.AutomaticIndexer.net/Folder#">
<rdf:Description rdf:about="http://www.AutomaticIndexer.net/Folder#code">
<Folder:dir>
    <rdf:Bag>
        <rdf:li resource="http://www.AutomaticIndexer.net/Folder#java"/>
    </rdf:Bag>
</Folder:dir>
<Folder:file>
    <rdf:Bag>
        <rdf:li resource="http://www.AutomaticIndexer.net/Folder#Readme.txt"/>
        <rdf:li resource="http://www.AutomaticIndexer.net/Folder#Help.txt"/>
    </rdf:Bag>
</Folder:file>
<Folder:name>
    <rdf:Bag>
        <rdf:li>code</rdf:li>
    </rdf:Bag>
</Folder:name>
</rdf:Description>
<rdf:Description rdf:about="http://www.AutomaticIndexer.net/Folder#java">
<Folder:name>
    <rdf:Bag>
        <rdf:li>java</rdf:li>
        <rdf:li>java2</rdf:li>
    </rdf:Bag>
</Folder:name>
<Folder:status>valid</Folder:status>
</rdf:Description>
```

Figure 3: View of the index.rdf file

3.4 Hash Index Table

The hash index table can be an in memory object or a database table that functions as a fast lookup. Without a fast lookup all the file requests are resolved via navigating the RDF graph. This consumes time and space since it is required to construct the RDF graph before a search can be performed. A fast lookup is used to avoid this performance bottleneck. This lookup can be implemented using in memory object or as a database table. The table traces all the changes to the file system by hashing the old reference and linking it with the new reference enabling file retrieval through the old reference. Figure 4 represents a view of a hash index table.

Key	Path	Order	Status
4d437f09531ae495b24ca85c2a284460	C:\code\java	1	Valid
4d437f09531ae495b24ca85c2a284460	C:\code\java2	2	Valid
84cf254d180ebc7f7e2f1393a12f2105	C:\code\Readme.txt	1	Valid
c48a3a704f6678c8da300810b5d671fa	C:\code\Help.doc	1	Deleted

Figure 4: View of the hash index table

For instance an index entry to C:\code\java folder changes to C:\code\java2. An entry is added to the hash index table which contains the same hash value to both entries. Further the order field will determine the latest update. In case the hash index table is empty or does not exist, file requests are resolved via navigating the RDF graph and then updating the hash index table.

3.5 Query Interface

There are two main uses of the Query Interface component. Firstly, the interface generates a master index using the sub-indexes stored in the individual directories for reference of applications. The master index is in RDF and is readily available for World Wide Web applications to use. The index is based on file names stored in the file system and acts as the central index.

Secondly, the component intercepts file retrieval requests from applications and facilitates index lookup to locate the stored file in the file system. To locate, the file Query interface generates a query to Path Builder component to dispatch the application request. The Path Builder uses the query as the entry point to locate the requested file.

3.6 Path Builder

The Path Builder component receives a search query from the Query Interface component and is used by the Path Builder component to validate and locate the file stored in the file system. The Path Builder locates the file. The component validates the existence of the file in the path. If the path exists the component tries to bind the file and returns a handler back to the application. In case the file name cannot be found the Path Builder initiates resolving the file name; firstly via the hash index table and secondly using the sub-index stored in the leaf node of the path. There are scenarios where the directory names in the referenced path have changed. The Path Builder resolves the directory names recursively tracking and binding name changes from root node to leaf node.

The Path Builder plays a vital role in the framework by resolving file and directory names to facilitate retrieval of files in a timely manner. The time it takes to retrieve a file, if it does not exist, is directly proportional to the depth of the tree (O(n)).

4. IMPLEMENTATION

The Automatic File Indexing Framework is implemented as a Windows service since the framework is required to monitor the file system changes throughout the span of the system operation. The service runs all the time monitoring file system continuously. The framework is implemented using Java language. A running prototype is completed and it requires performance fine tuning before a final release. The service is created and launched by using the Java Service Wrapper, an open project.

To monitor the file changes the java.nio.file package which provides a file change notification API, called the watch service API is used to implement file system monitor component. When the watch service detects file notification events they are forwarded to the main thread which allocates a worker thread from the thread pool. The thread pool size is determined heuristically. The worker threads processes any event registered in the main thread. Figure 5 represents the code used to detect the registered events.

```
if(event.kind()==ENTRY_MODIFY)
{
    z++;
    newName=child.toString();
    modifyNode(event,newName);
}
else if(event.kind()==ENTRY_CREATE)
{
    x++;
    newName=child.toString();
    addNode(event,newName);
}
else if(event.kind()==ENTRY_DELETE)
{
    y++;
    oldName=child.toString();
    deleteNode(event,oldName);
}
if (recursive && (kind == ENTRY_CREATE)) {
    try {
        if (Files.isDirectory(child, NOFOLLOW_LINKS)) {
            registerAll(child);
        }
    } catch (IOException xg) {
    }
}
```

Figure 5: Handling registered events

Partial indexes are created and searched using Jena RDF API. Jena is a Java API that facilitates creation, manipulation & navigation of RDF graphs. The API is used to maintain index files at each folder level called partial indexes as depicted in figure 2. The main flow of the algorithm for serving a file request is shown in figure 6.

The algorithm is implemented by using the sub-components explained in section 3.

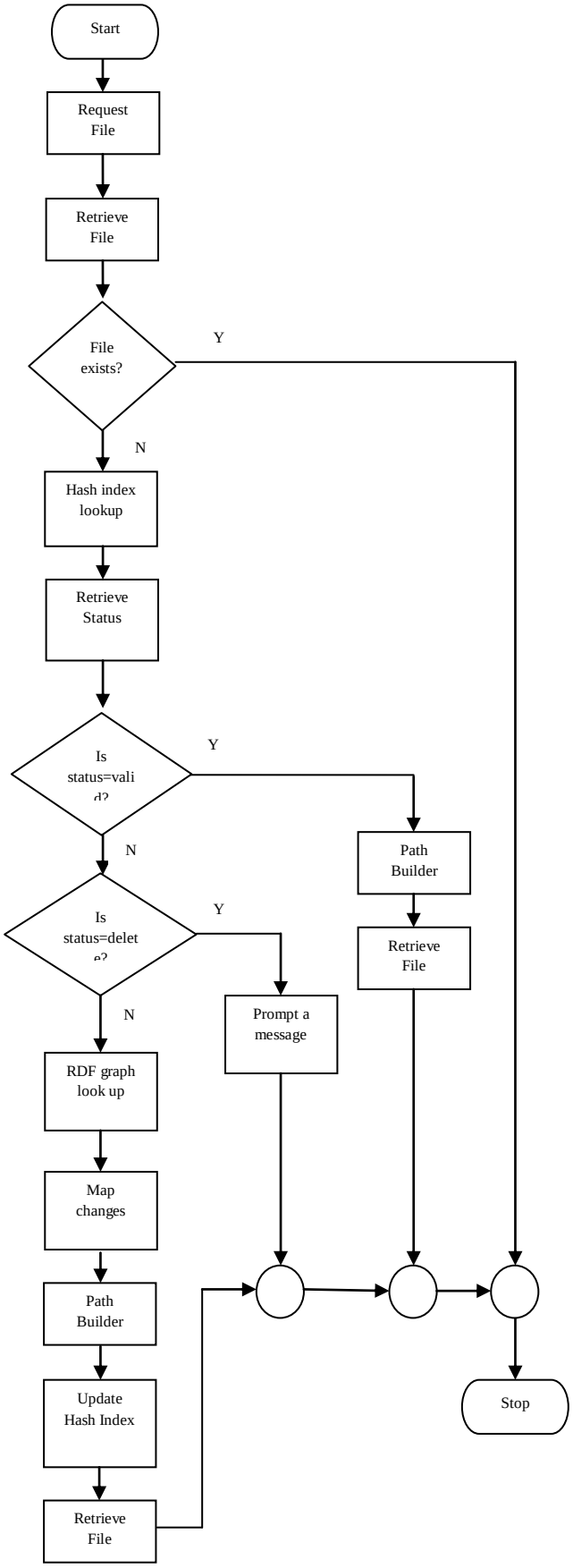

Figure 6: Main flow of a file request

5. RESULT ANALYSIS

Existing document indexing software tools focus on building indices using data structures to locate files based on specific parts of text found in them. The research domain is deeply rooted to index structures ranging from most simple look-up tables to sophisticated graph-based index structures for search queries [10]. The process is more or less similar to the process of using an index of a book. For more recent development, the tendency is to find what the User wants through a file search interface where it recommends and retrieves similar files based on how far they are close to User's search text.

The systems that exist in the market use a stored index or an index is built on demand. In both scenarios a User has to be in the loop to find what he or she wants. Most of the software tools according to a recent review of best document management software packages [11] reveal that most of them support common features namely full-Text search, backup & restore, central repository, multiple versions support and the like. The solutions that exist are less related to the proposed work. In brief the Automatic File Indexing Framework is an automatic indexing framework which resolves dangling file pointers by maintaining sub-indexes at each folder level that track changes to the file system. The tracking is done in the background. It makes the index intelligent by locating the original file irrespective of all the revisions done and without an involvement by the User. The folder level descriptors are in RDF files that can be easily integrated to semantic indexing. Therefore, the index is readily available for the Web. Current software packages in the market do not support these features.

6. SUMMARY AND FUTURE WORK

In this paper an Automatic File Indexing Framework, which generates sub-indexes automatically at the folder level was presented. The sub-index is a RDF file and it uses Resource Description Framework to maintain it. The RDF file acts as a metadata file that stores directory changes locally. Further, the RDF file tracks and records changes done to the directory entries automatically. This makes the sub-index intelligent when resolving the names later. The framework receives on demand file retrieval request from applications and is able to return the file back to the application irrespective of the directory changes and file name changes. This makes the framework unique in solving dangling file pointers.

Currently, a working prototype is implemented to prove the proposed framework solves the existing problems effectively. It is developed using JAVA platform. The system uses Jena RDF API [12] & file change notification API. The Jena RDF API is used to build and locate sub-indexes stored in the file system while file change notification API is used to monitor the file system for changes continuously. According to the initial test runs the results were promising with respect to solving dangling pointer to files but the prototype requires performance tuning to suit large file repositories.

The Automatic File Indexing Framework is an effective solution to dangling file pointes. The indexes can be further improved by storing summarized file contents to enable a semantic index search. It will lift indexing and searching of the applications. Another area for improvement is to provide a file recommender system for the users where the system recommends files based on their similarity to the structure and content. So far, the prototype implemented solves the problem of dangling file pointers and preparations are underway to systematically evaluate user responses to the framework. The framework gives an experience that a user has never experienced before in locating what was wanted.

7. REFERENCES

[1] Google Desktop Search. Available http://desktop.google.com/en/ (accessed 11 January 2011).

[2] Saved Search File Format. Available http://msdn.microsoft.com/en-us/library/bb892885 (v=vs.85).aspx (accessed 20 January 2011).

[3] Windows Search. Available http://windows.microsoft.com/en-US/windows7/ products/features/windows-search (accessed 12 January 2011).

[4] Dumais, S.T., Cutrell, E., Cadiz, J., Jancke, G., Sarin,R., Robbins, D.C., "Stuff I've Seen: A system for personal information retrieval and re-use", ACM SIGIR, 2003.

[5] Darren R. Hardy, Michael F. Schwartz, "Essence: A Resource Discovery System Based on Semantic File Indexing", In USENIX Winter, pp. 361-374, 1993.

[6] Chang K., Perdana I.W.T., Ramadhana B., Sethuraman K.,Le T.V. and Chachra N "Knowledge File System - A Principled Approach to Personal Information Management.", ICDM Workshops IEEE Computer Society, pp. 1037-1044, 2010.

[7] Gemmell J., Bell G. and Lueder R., "MyLifeBits: a personal database for everything", Communications of the ACM, vol. 49, Issue 1, pp. 88-95, Jan 2006.

[8] Mourra, John (Toronto, CA), Klicnik, Vladimir (Oshawa, CA), Loi, Lok Tin (Toronto, CA), Tsuji, Hiroshi (Stouffville, CA), "File indexing framework and symbolic name maintenance framework", United States Patent 7873625, 2011

[9] Resource Description Framework (RDF). Available http://www.w3.org/RDF/ (accessed 2 February 2011).

[10] Weigel, F. 2002 A Survey of Indexing Techniques for Semistructured Documents. Journal of Ludwig Maximilians. Universitat Munchen.

[11] Document Management Software Review 2012. Available http://document-management-software-review.toptenreviews.com (accessed 21 July 2012).

[12] An Introduction to RDF and the Jena RDF API. Available http://jena.sourceforge.net/tutorial/RDF_API/ (accessed 2 February 2011).