Search, Versioning, and Temporal Search for Desktops

Keishi Tajima
Kyoto University
Tajima@i.kyoto-u.ac.jp

1. Project Goal

Nowadays, many knowledge workers or even non-expert computer users store large amount of data files on their computers, and various functions to help users managing those files efficiently have become an important function of operating systems or desktop environments in order to improve the productivity of the people in the world. Among such functions for file management, in this research project, we especially focus on the functions for searching for files, and the functions for managing versions of files.

Functions for searching ones' desktop for files, i.e., desktop search, have already become very important features of many operating systems. In the existing desktop search systems, however, user can only specify keywords and conditions on some meta data, such as file creation time. Another useful information in desktop search is information on structure inside data files. While in Web search, users mainly want to find new unknown Web pages, in desktop search, users mainly want to retrieve some file they saw before. Therefore, fragmentary knowledge on their structure in their memory is very useful in desktop search. In this project, we develop a desktop search system that allows users to specify such fragmentary information on structure.

To retrieve files including the given structure, we need to be able to extract structure inside various kind of data files on the desktops. The simplest approach is to provide a wrapper or parser for each file type, which extracts structure in that file type into some universal format. The cost of providing such a wrapper for every file type is, however, too high, and such a approach can easily lead to the situation where structural query may fail to retrieve some files of file types that do not have their wrappers, and users hesitate to specify structural information in their queries. To solve this problem, one goal of this research project is to design a framework where wrappers can be defined as easily as possible, and users can retrieve files even when its file type does not have its wrapper.

The second topic is the management of versions of files. One fundamental issue in version management on desktops is how to find files corresponding to versions of the same file. Users know which file is a new version of which file, but it is not desirable to have users to specify it every time some file is newly created including files automatically created by some software. Therefore, we need some mechanism to automatically identify which file is a new version of which file.

In many version management systems used for software development, file names are keys to identify "same" files. In the desktop environments, however, file names are not always keys because we sometime use different file names to represent versions of one "same" file, e.g., report090530.doc and report090531.doc. There are also many unrelated files that happen to have the same file name but in different directories. Note that there are also files representing versions of the same file and are stored in different directories. Many operating systems have some form of file IDs, but those IDs cannot be keys either because we often reuse some existing file and edit it into another new file. The log of the user operations, such as copy operations, is not useful for the same reason. In this way, how to automatically identify which files are versions of the same file is not a simple problem, and this is the second issue to study in this project.

2. Technical breakthrough

One important observation in the structure-based file search on the desktops is we usually do not need the complete view of structure inside files. The users rarely remember and want to specify the detailed structure of files they need. They usually remember and specify only fragmentary information. Therefore, it is not necessary to create high-cost perfect wrappers, and simple wrappers only extracting fragmentary important structure in the best-effort fashion suffice in most cases.

Another observation is there is a spectrum ranging from file types for which it is hard to write wrappers, e.g. binary file types whose detailed format information is not disclosed, to file types for which it is relatively easy to extract the main structure, e.g., text-based format including clear data delimiters, or even file types for which its perfect structural view is available, e.g., recent XML-based office document formats.

Based on these observations, we designed a framework where we can easily define rules to extract structure from various file types. In our framework, we first classify file types into four categories:

Cat. 1: file types for which we provide wrappers that extract a list of text,
Cat. 2: file types for which we provide wrappers that extract a list of text and delimiters that can decompose the list into sub-lists,
Cat. 3: file types for which we provide wrappers that extract a list of attribute-value pairs, and
Cat. 4: file types for which we provide wrappers that extract a labeled tree structure.

Then, given a user query which includes query keywords and may or may not include structural information, we transform that query to four queries each of which is for each category above. We evaluate each of those four queries over the files in the corresponding categories, and then merge the four results into one final query answer set.

To implement such a framework, we designed a set of translation rules that translate a given query into an appropriate query for each category with preserving as much semantics in the original query as possible in the translated query. We omit the details, but we show an example of the translation in Fig. 1.

Cat 4	// * [. // date = "*2008*"][. // * = "*Beijing*"]
Cat 3	(date: *2008*, *: *Beijing*)
Cat 2	{ date , 2008} , Beijing
Cat 1	date, 2008, Beijing

*Fig.1: Translation of the query "date: 2008, *: Beijing"*

For the automatic identifications of files representing versions of the same file, we also designed a framework where we can specify "rules" for such identification. Rules define when we regard two files as the versions of the "same" file. For example, we can define a rule that says if two .doc files have filenames that match the pattern *2009????.doc with the same "*" part, we regard them as the two versions of the same document. Similarly, we can define a rule that says two .eml files (storing an email message) are the same file only if they have the same Message-ID line in their header parts even if they have different file name and are stored in different folders. By defining such rules for each file type, we can have the system automatically find the pair of files representing versions of the same file.

3. Innovative Applications

By using our framework for structural file search, users can issue a desktop query like "date: 2008, *:Beijing", which means files including attribute whose attribute name includes "date" and whose value includes "2008", and also including keyword "Beijing" somewhere. Then, our system translate this query into four queries corresponding the four categories as shown in Fig.1, and may retrieve email files including the text "2008" in its date header and including the word "Beijing" somewhere, and also retrieve any files classified in Cat.1 that include text "date", "2008", and "Beijing" somewhere (Fig.2). Note that we do not require files in Cat.1 to include "date" as an attribute name because we do not provide a wrapper that provide such information for Cat.1 files.

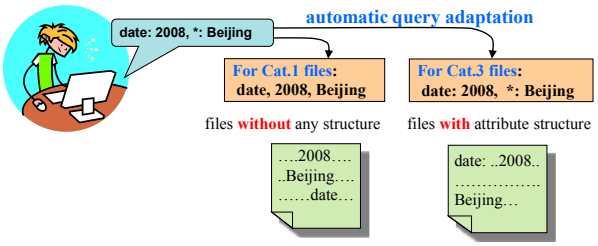

Fig.2: Automatic adaptation of structural queries to various file type categories

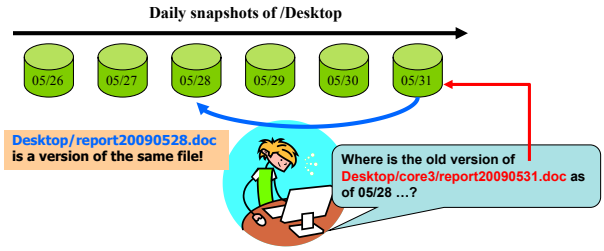

Fig.3: Identification of files representing different version of the same document

Our framework for identifying versions of a file can be applied to temporal desktop search systems. Suppose, we archive the daily snapshot of whole directory structure of some users' desktop. One day, a user want to retrieve an old version of his file "Desktop/grant/core3/report090531.doc" as of last Friday. Then, the system searches the snapshot of the desktop on last Friday, and identifies the file "Desktop/report090526.doc" is the old version of that file (Fig.3).

4. Academic Achievement

We are now implementing our proposed systems and preparing the papers for publication.

5. Project Development

The project is on going (without a support by a particular grant).

6. Publications

Paper publication

In preparation.