

# Improving Precision of Office Document Search by Metadata-based Queries

**Somchai Chatvichienchai**

Dept. of Info-Media, Siebold University of Nagasaki,  
1-1-1 Manabino, Nagayo Cho, Nishisonogi Gun, Nagasaki, 851-2195, Japan.  
E-mail: somchaic@sun.ac.jp

and

**Katsumi Tanaka**

Dept. of Social Informatics, Kyoto University,  
Yoshida, Honmachi, Kyoto 606-8501, Japan.  
E-mail: ktanaka@i.kyoto-u.ac.jp

## **Abstract**

*As office documents created by office applications are shared among users in the enterprises, finding the documents that fit users' needs from disparate data sources is an essential issue. Search methodology of conventional desktop search tools is based on keyword occurrences in the documents. Then, these tools potentially return documents that are not relevant or miss documents even though they are relevant. This paper shows how free-text queries of conventional search engines can be enhanced with metadata describing document types and field names in order to increase precision of office document search. A technique that extracts fields and metadata from office documents and generates a search index which allows users finding out office documents by the enhanced search queries is proposed.*

*Keywords: Metadata, documents, XML, search index, query.*

## **1. Introduction**

Office applications, such as Microsoft Office Suites[8] and OpenOffice[9] etc, are widely used in modern enterprises because they have simple, easy-to-use interfaces and contain full-featured applications including word processing, spreadsheet, presentation and graphics. As more and more office documents created by these applications are shared among users in the enterprises, finding the documents from disparate data sources that fit users' needs is an essential issue. Recently a number of desktop search tools [6, 13], which were released by major search engine vendors (Google, Yahoo etc.), are used to search files in user computers. These tools build and maintain an index database to achieve reasonable performance when searching several gigabytes of data. However, these desktop search tools lack a mean that allows users to define which fields of the documents the search tools should compare with terms of search queries and names of these fields. Therefore, these tools can't allow user to define search queries that indicate only relevant documents. Consider a sample of a purchase order (see Fig.1) created by a spreadsheet program. Suppose that Tanaka wants to find out the purchase

Qty	Product No.	Description	Unit Price	Amount (Yen)
30	101001	Mild Green Tea 300g	3,200	96,000
40	102002	Herb Tea 250g	2,800	112,000
Subtotal				208,000
Sales Tax (5%)				10,400
Total				218,400

Fig.1: A sample document created by MS Excel.

orders whose delivery dates are before 2007/8/31. Since conventional search tools can't identify which field ("2007/8/20" or "2007/9/10") is the delivery date, these tools can't correctly decide whether this document is one of the documents that Tanaka wants. Therefore, this example shows that these tools potentially return documents that are not relevant or miss documents even though they are relevant.

The objective of this paper is to propose a technique that enhances free-text queries of conventional search engines with metadata describing document types and field names in order to increase precision of office document search. A technique that collects fields and metadata from office documents, and generates a search index that allows users searching target documents by the enhanced queries is also presented. This paper focuses on the documents that are edited by Microsoft Office Word 2003/2007 and Excel 2003/2007 since these office applications are used in many enterprises and public organizations. Therefore, the term "documents" in this paper denotes the digital documents edited by these office applications.

**Related Work:** Semantically enhanced search was addressed from other perspectives, as in *Stuff I've Seen* [4], where contextual terms (e.g., access time, or author) are used

to enrich search results, or as in *Swoogle* [2], in which information retrieval capabilities are offered for semantic documents residing on the Web. [3] shows how search technology can be enhanced with implicit predicates, in order to take into account the structure and semantics defined by applications.

## 2. Background

In this section some basic Information Retrieval (IR) concepts and XML terminology are given as follow.

**Free-text Queries.** Most IR systems support free-text queries, allowing Boolean expressions on keywords and phrase searches. Support for mandatory and forbidden terms is also common, e.g. the query *+coffee sugar -cream* indicates that *coffee* is mandatory, *sugar* is optional, and *cream* is forbidden. Most systems also support fielded search terms, i.e. terms that should appear within the context of a specific field of a document, e.g. *+author:Somchai author:Tanaka*.

**Inverted List.** Most IR systems use inverted lists as their main data structure for full-text indexing [10]. There are many literatures on efficient ways to build inverted lists (See e.g. [1, 5]) and evaluate full-text queries using them (See e.g. [7]).

**XML (eXtensible Markup Language).** XML[12] provides a way to describe structured data. An *XML document* is a plain-text file that uses XML tags to define the logical structure of the document in a hierarchical fashion. XML uses a set of tags to delineate elements of data. An element contains subelements and attributes. These subelements in turn contain other subelements, and so on. XML documents can be abstracted as labelled trees (called *XML trees*). XML schemas [11] are documents

that are used to define and validate the content and structure of XML documents. XML schemas can be abstracted as labelled trees (called *schema trees*)

### 3. Overview of the Proposed Technique

In order to enable users to define search conditions that can effectively identify the target documents, conventional free-text queries are extended to include search conditions of the following format:

*fieldDefinition*:*[operator]*"*value*",

where *fieldDefinition* is a field definition which is presented by *doctype::fieldname* (where *doctype* denotes document type; and *fieldname* denotes name of a field of document of *doctype*); *operator* denotes "=", ">", "<", ">=" or "<="; and *value* denotes the keyword or phrase that should appear in the specified field.

#### Example 1:

*PO::deliveryDate:[<=]"2007/09/31"* denotes conditions for finding the documents of *PO* type whose text strings appearing in *deliveryDate* fields are less than or equal to "2007/09/31".

A search index for office documents is generated by the following steps.

**Template Design.** In order to decrease overhead in creating and maintaining documents, many enterprises employ templates to unify formats of documents of the same type. For each document type, *system manager* (who is the administrator of the proposed document searching system) creates a template which contains *dummy data fields* which will be replaced by the exact data when users create a new document.

**Field-Schema Definition.** For each template, system manager creates a *field-schema* that

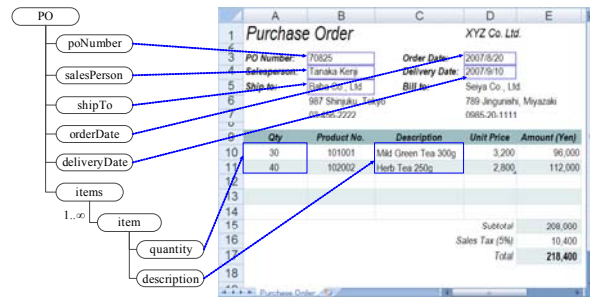


Fig.2: A sample spreadsheet whose cells are mapped to elements of field-schema of purchase orders.

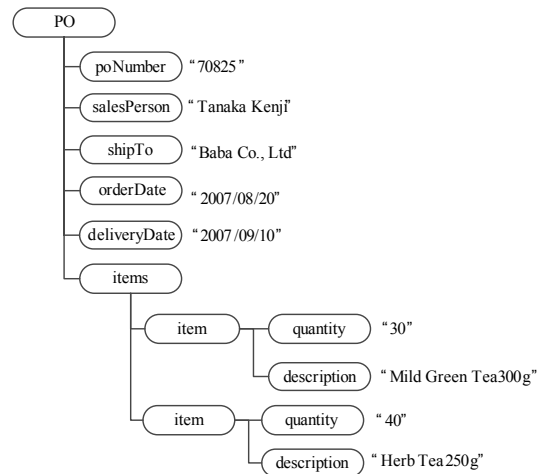


Fig.3: A sample XML tree of a field-schema instance file.

defines the fields that the system should extract from the documents created from the template in order to generate the search index. System manager defines locations of the fields in (Word or Excel) documents of a template by creating a mapping from all or some of elements of the corresponding field-schema to dummy data fields of the template thru the XML map function (of Word or Excel, respectively). Figure 2 depicts a sample spreadsheet whose cells are mapped to some elements of field-schema of purchase orders.

System manager embeds a macro program to the template and make it activate when the document created from the template is saved. When users save the document, this macro automatically outputs the mapped fields into a field-schema

instance file. Figure 3 illustrates a sample XML tree of a field-schema instance file generated when the document of Fig.2 is saved.

**Index Creation.** The proposed index is a set of inverted lists having the follow format.

$\langle docid, field, fieldDefinition, order \rangle$ ,

where *docid* is arbitrary information that identifies a document file; *field* is a field inside the document identified by *docid*; *fieldDefinition* denotes a field definition of *field*; and *order* is order number of *field* appearing in the document. Note that *fieldDefinition* is presented by *doctype::fieldname*, where *doctype* denotes document type; and *fieldname* denotes name of a field of document of *doctype*.

Given a field-schema instance file, index creation program generates its XML tree and traverses nodes of the XML tree from the root in preorder. For each leaf node of the XML tree, the program generated an inverted list whose data fields are determined by the following criteria.

- *docid* is defined by a unique pointer which locates memory address storing file path of the corresponding office document.
- *field* is defined by text values of the current node.
- *fieldDefinition* is basically defined by the concatenation of element name of the root node and “::” and element name of the current node.
- *order* is basically defined by the number of nodes (that have been traversed prior the current node and are belong to the same element as the current node) plus one. However, definition of *order* may vary on the document type.

**Example 2:** Inverted list  $\langle d1, Herb Tea 250g, PO::description, 2 \rangle$  of the search index file

docID	field	fieldDefinition	order
d1	70825	PO::poNumber	1
d1	2007/08/20	PO::orderDate	1
d1	2007/09/10	PO::deliveryDate	1
d1	Tanaka Kenji	PO::salesPerson	1
d1	Baba Co., Ltd	PO::shipTo	1
d1	30	PO::quantity	1
d1	40	PO::quantity	2
d1	Mild Green Tea 200g	PO::description	1
d1	Herb Tea 250g	PO::description	2
d2	85119	PO::poNumber	1
d2	2007/09/05	PO::orderDate	1
d2	2007/09/25	PO::deliveryDate	1
d2	Kato Ichiro	PO::salesPerson	1
d2	P&K Company	PO::shipTo	1
d2	50	PO::quantity	1
d2	Aroma Coffee 400g	PO::description	1

Fig.4: A sample search index file.

shown in Fig.4 states that “Herb Tea 250g” appears as the second field of *PO::description* of document *d1*. The combination of  $\langle d1, Herb Tea 250g, PO::description, 2 \rangle$  and  $\langle d1, 40, PO::quantity, 2 \rangle$  states that the quantity of *Herb Tea 250g* of document *d1* is 40.

#### 4. Document Searching

At query time, the search index is used to find matching documents for a search query. Basically a search query consists of mandatory, optional and forbidden conditions.

Let  $q = \{m_1, \dots, m_p, o_1, \dots, o_r, f_1, \dots, f_s\}$  be a search query where  $m_i$  ( $1 \leq i \leq p$ ) is a mandatory condition,  $o_j$  ( $1 \leq j \leq r$ ) is an optional condition, and  $f_k$  ( $1 \leq k \leq s$ ) is a forbidden condition. Let  $D(c)$  be the set of document IDs of the inverted lists satisfied by condition  $c$ . Let  $M(q)$ ,  $O(q)$  and  $F(q)$  be the query results of mandatory, optional and forbidden conditions of query  $q$ , respectively. The result of query  $q$  is processed by the following steps.

1. Find the query result of conditions of the same type which is defined as follows.

$$M(q) = \bigcap_{i=1}^p D(m_i) \quad \text{where } \bigcap \text{ denotes the intersection operator.}$$

$O(q) = \bigcup_{i=1}^r D(o_i)$  where  $\bigcup$  denotes the union operator.

$F(q) = \bigcup_{i=1}^s D(f_i)$

- The result of query  $q$  is  $(M(q) \cup O(q)) - F(q)$ .

**Example 3:** A search query for the purchase orders which have been planned to deliver “Mild Green Tea 200g” before 2007/09/31 can be defined as follows:

`+PO::description:[=]“Mild Green Tea 200g”`

`+PO::deliveryDate:[<=]“2007/09/31”`.

Based on the sample search index file of Fig.4, the set of document IDs of the lists satisfied by `+PO::description:[=]“Mild Green Tea 200g”` is  $\{d1\}$ . The set of document IDs of the lists satisfied by `+PO::deliveryDate:[<=] “2007/09/31”` is  $\{d1, d2\}$ . Since  $\{d1\}$  is the result of intersection of  $\{d1, d2\}$  and  $\{d1\}$ , the result of the above query is  $\{d1\}$ .

## 5. Conclusion

As conventional desktop search tools lack a mean that allows users to define which fields of files that should be compared with terms of search queries, these tools can't allow user to define sufficient conditions of search queries whose results indicate only relevant documents. In order to solve this problem, this paper has enhanced conventional free-text queries with metadata describing document types and field names of office documents. A technique that collects fields and the metadata from office documents, and generates a search index has been proposed. The paper has also presented a method of computing query result from the search index.

## References

- [1] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In WWW'98, p.107–117, 1998.
- [2] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. C. Doshi, and J. Sachs. Swoogle: A search and metadata engine for the semantic web. In CIKM'04, p.652-659, 2004.
- [3] J.-P. Dittrich, C. Duda, B. Jarisch, D. Kossmann, M.A. Vaz Salles ETH Zurich: Bringing Precision to Desktop Search: A Predicate-based Desktop Search Architecture. In ICDE'07, p.1461-1465, 2007.
- [4] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff i've seen: A system for personal information retrieval and re-use. In SIGIR'03, p.72-79, 2003.
- [5] M. Fontoura, E. J. Shekita, J. Y. Zien, S. Rajagopalan, and A. Neumann. High performance index build algorithms for intranet search engines. In VLDB'04, p.1158–1169, 2004.
- [6] Google. 2006. “Google Desktop Search,” <http://desktop.google.com/>.
- [7] X. Long and T. Suel. Optimized query execution in large search engines with global page ordering. In VLDB'03, p.129–140, 2003.
- [8] Microsoft. 2006. Microsoft Office Suites. Available at <http://office.microsoft.com/en-us/suites/FX101677751033.aspx>.
- [9] OpenOffice. 2007. OpenOffice Suite Version 2.1. Available at <http://www.openoffice.org/index.html>.
- [10] I. Witten, A. Moffat, and T. Bell. Managing Gigabytes. Morgan Kaufmann, 1999.
- [11] W3C. 2001. “XML Schema,” Available at <http://www.w3c.org/XML/Schema>.
- [12] W3C. 2006. “Extensible Markup Language (XML) 1.0 (Fourth Edition),” Available at <http://www.w3.org/TR/2006/REC-xml-20060816/>.
- [13] Yahoo. 2006. Yahoo Desktop Search, <http://desktop.yahoo.com/>.