Context Driven XML Retrieval
Aneliya Tincheva
IPP-BAS
25A Acad. G. Bonchev Str.
Sofia 1113
nelitincheva@gmail.com

Abstract
This paper presents a data-centric approach to XML information retrieval which benefits from XML document structure and adapts traditional text-centric information retrieval techniques to deal with text content inside XML. We implement our ideas in a configurable, general purpose XML retrieval library which can be tuned to operate on multilingual XML resources with different structure and can be used to extract relevant document fragments with different granularity according to user preferences. We present a rich query format and an algorithm for indexing and query processing.

Keywords
XML Retrieval, IR, XML-IR, XPath, document fragment, indexing schema, full-text indexing

1. Introduction
The popularity of the eXtensible Markup Language (XML) has led large quantities of structured information to be stored in this format. Due to this ubiquity, there has lately been interest in information retrieval (IR) from XML. XML-IR presents different challenges than retrieval in text documents due to the semi-structured nature of the data. The goal is to take advantage of the structure of explicitly marked up documents to provide more focused retrieval results. For example, the correct result for a search query might not be a whole document, but a document fragment. Alternatively, the user could directly specify conditions to limit the scope of search to specific XML nodes. Previous work [2, 4] addresses several challenges specific to retrieval from XML documents:
(1) Granularity of indexing units (Which parts of an XML document should we index?)
(2) Granularity of the retrieved results (Which XML nodes are most relevant?)
(3) Ranking of XML sub-trees (How should the ranking depend on the type of enclosing XML element and term frequency/inverse document frequency (tf-idf)?)

The aim of this work is to define an approach for XML retrieval that can be used for indexing and search independently of the document structure. We call our approach context driven XML retrieval because indexing and search operate on parts of XML documents called contexts. These contexts represent searchable and retrievable parts of an XML document, and for us the IR problem can be viewed as the extraction of contexts that match some search criteria. Traditional IR is a special case of XML-IR where the context has to be a whole document. Narrower contexts could be separate XML elements or their combinations. Our setting assumes knowledge of the XML document structure and the retrieval requirements. Thus an administrator creates indexing and retrieval rules for different XML document corpora. Each corpus requires different indexing rules to define contexts and relations between them – document fragments referable at search time. Using this context driven approach we address challenges (1) and (2).

Concerning ranking (3), we employ a strategy which combines the unstructured and structured IR scoring techniques. In the paper we present a scalable index structure, indexing, search algorithms, indexing rules and query language format. We implement our ideas in a general purpose XML retrieval library that can be integrated in different kinds of applications: web applications, standalone systems, web services.

The rest of the paper is organized as follows: Section 2 describes related work; Section 3 describes motivation; Section 4 presents implementation details. Section 5 concludes the paper and describes future work.

2. Related work
XML retrieval systems vary according to the query language, index structure, document preprocessing, indexing and scoring algorithms they employ. A great variety of XML query languages already exist. Standard ones proposed by W3C are XPath and XQuery. Unfortunately, they do not reflect IR properties such as weighing, relevance-oriented search, data types and vague predicates, structural relativism [1, 14]. Amer-Yahia et al. [4] classifies XML query languages into three classes: keyword query languages (KQL) [5, 6, 12, 13]; tag & KQL [6]; path & KQL [7, 12]; XQuery & KQL [10].

The query language we introduce is a path and KQL in XML format, and most related to XPath 2.0, XIRQL, XXL, NEXI CAS queries. Different term and structure statistics are implemented in separate XML-IR systems. We share the idea of Mass and Mandelbrod [11] that an XML index consists of a set of separate full-text indices. For full-text search we use the Apache search API Lucene [9].
The context driven approach we present can be classified as Content-And-Structure (CAS) retrieval under the system developed by the Initiative for the Evaluation of XML Retrieval (INEX) [12].

3. Motivation

In addition to the growing interest in XML retrieval, we had a practical need for an IR system for XML documents. In order to aid annotation efforts, we needed a platform independent search engine that could be tuned for specific applications. Since the document structure was known and important, we wanted to create indexing and retrieval rules to improve retrieval. The system we present allows exactly such application-specific indexing and search.

4. Context Driven XML Retrieval

An XML document is a tree-like data structure which consists of three node types: elements, attributes, character data/text. XML document tree nodes are instances of elements called tags/markups, which can be either empty or have nested elements or text nodes. Attributes are name-value pairs attached to tags. Figure 1 is an example of a textile multimedia XML document created by our group for the purposes of the AsIsKnown project [15]. The example document contains text and images annotated with concepts from a textile knowledge base. Text is delimited in sentences which are organized in paragraphs.

Figure 1. Multimedia XML Document

There exist W3C standard languages for navigation through XML documents (XPath) and querying (XQuery). We employ XPath in the implementation of our framework. After the evaluation of an XPath expression, a set of XML nodes are retrieved. For example, if we want to extract the sentences which contain the word pattern from our example XML document in Figure 1, we can use the XPath expression: 

```
//s[contains(descendant::text(), "pattern")]
```

We set the variable $x$ to be equal to the document title text. The XPath engine extracts the sentences whose text contains the value of the $x$ variable.

4.1 System architecture

We implement our system in an XML retrieval library. Each document corpus has a separate XML index in a central XML Index Repository. Each index has its own model, defined in an indexing schema. The indexing schemas specify how to extract indexing units (XML sub-trees) called contexts from XML documents with a common structure and defines how the separate contexts are related. Our basic assumption is that a context is a document fragment whose content is indexed in several text fields. A field is a name-value pair whose value is character data. Contexts and fields are indexed in search queries. An indexing schema is added to an empty index on its creation. Existing XML indices are populated with documents according to the definitions in their corresponding indexing schema. For each context defined in the indexing schema we create and populate a full-text index called context index. The rest of the subsection gives an overview on the system architecture (Figure 2).

Figure 2. System Architecture

The search engine lifecycle has two sequential phases: system initialization and user interactions. When the system is started a processing module reads system configurations from a configuration file. This file specifies an indexing/search analyzer, document storage, and result extractor implementation classes. The opportunity to configure different implementations makes our framework highly adaptable to various search scenarios. We can use this to tune text analysis, document access policy, result formatting and extraction. Instances of the configured implementations are created by reflection and are made
available at runtime. The indexing and retrieval tasks are performed by the indexer and searcher operational modules which manipulate the XML indices, system files and interact with the already instantiated objects.

4.2 Indexing schema and index structure

Each XML index consists of an indexing schema and one or more full-text indices. The indexing schema is an XML document which defines indexing and extraction rules. Central concepts are context and field, as defined in the previous subsection. Their usage is clarified with the following example. Assume that we have a corpus with documents with the same structure as the one in Figure 1 and we want to retrieve particular paragraphs/images. For the purpose we define 2 independent contexts: paragraph and image. Our aim is to search for a combination of text matches and concepts. We need to create two fields for the paragraph context (text and concept) and one for the image context (concept). We add one more requirement - we want to extract paragraphs whose adjacent paragraphs and images comply with supplementary search criteria. In this case we need to define some kind of relation between a paragraph and its adjacent paragraphs and images. We call this type of relation coordination between contexts. In the indexing schema the contexts are defined as context elements, fields are elements nested in context elements and coordination between contexts is expressed by nesting context elements.

Figure 4. Indexing Schema Logical Tree View.

Each context and field element has an identifier and is associated with an XPath expression. Indexing schema elements are relative to their parent elements, i.e. their identifiers are unique within the scope of their parent element and their XPath expressions are applied relative to the nodes extracted by the XPath expressions of their parent element. The schema root element denotes the default context, i.e. the whole XML document. Field elements have no nested elements and can be boosted. For context elements with child field elements we create separate full-text indices in the XML index repository. Figure 5 illustrates the Lucene index (full-text index) and the XML index structures.

Figure 5. Lucene and XML Index Structure

The Lucene index (left) is an inverted index consisting of a set of Lucene documents. Each Lucene document carries a unique identifier and contains fields. The XML index (on the right) contains an indexing schema document and a set of Lucene context indices. They are populated with Lucene documents with identifiers which encode the system identifier of the XML document, path to an XML context node and paths to its related XML nodes. The Lucene documents are populated with fields as defined in the indexing schema document. An illustrative example is to index the document in Figure 1. We want to search by concept to extract paragraphs that match one concept pattern with adjacent paragraphs or images matching another pattern. Such a relation between structural document units allows us to create complex search queries. An indexing schema satisfying our requirements is given in Figure 6.

Figure 6. Example indexing schema

The index created according to the definitions of the indexing schema on Figure 6 contains five full-text indexes (one for each context with a field). If the document in Figure 1 has a system identifier fl, the content of the separate full-text indexes after the indexing would be:

| Lucene context index | Lucene document IDs | Field content |
|----------------------|--------------------|---------------|
| paragraph            | fl_p#1             | The concept URIs in the first paragraph. |
| paragraph ➔ previous_paragraph; (no data added)* | fl_p#1@@fl_img#1 | The concept URIs in the first image. |
| paragraph ➔ following_paragraph (no data added)* | fl_p#1@@fl_img#1 | The concept URIs in the first image. |
Table 1. Content of Lucene indices for the example XML Index

Indexing is incremental. When a new XML document is added to an XML index, its Lucene context indices are updated by creating and populating Lucene documents.

4.3 Indexing algorithm

Below (Figure 7) is listed the indexing algorithm which is recursive in nature. The recursive structure is inherited from the nesting of contexts.

| Indexing algorithm details |
|----------------------------|
| 1. Create an empty Lucene document |
| 2. For each field in the index: |
|   a. Create a new Lucene field |
| 3. Add the Lucene document to the index |

Figure 7. Search procedure

4.4 Query syntax and search algorithm

Search is performed within a single index in order to retrieve relevant content. The search criteria are specified in a query XML document whose format is presented below. Contexts and fields\(^2\) are referred to in search queries. Figure 8 (below) illustrates the query structure.

Queries have a recursive structure similar to the indexing schema. The context and field elements in queries refer to corresponding elements in the indexing schema by ID. The parent-child element relationship for both contexts and fields should follow the order in the indexing schema. The element content of query context elements consists of:

- **field element(s)** – their element content is transformed to a Lucene query. The supported full-text queries for a field include term, phrase, fuzzy, Boolean, span, wildcard, range queries. Search results are sorted by tf-idf.

- **AND**: OR \( \land \) and;or: elements – recursive multi-argument operations denoting intersection and union of search results returned for sets of context arguments (AND: OR) or field arguments (and; or). In either case, results are sorted by tf-idf.

The search algorithm is presented below (Figure 9).

![Figure 7: Search procedure](image)

**Figure 7. Search procedure**

**Figure 8. Search query structure**

\(^1\) The @@@ is a separator between identifiers of dependent contexts. The Lucene document identifier f1_p#1@@@f1_img#1 (row 4) encodes that the first paragraph is related to the first image with a previous_images relation.

\(^2\) Contexts and fields as defined in the indexing schema for the index in which we are searching.
the concepts in the preceding and following paragraphs and images are indexed in index \(I_2\). Finally, the text of paragraphs, besides the concepts, is indexed in index \(I_3\). For text analysis we used the Lucene \textit{StandardAnalyzer}.

This class uses a Java CC-based grammar to tokenize alpha-numeric strings, acronyms, company names, e-mail addresses, computer host names, numbers, words with an interior apostrophe, serial numbers, IP addresses, andCJK (Chinese Japanese Korean) characters. As we see in Chart 1, relatively little time is consumed in the creation of \(I_1\). Increasing the schema complexity degrades indexing performance.

This degradation was not a major concern. We focus on estimating:

1. Does indexing performance degrade drastically for complex indexing schemas?
2. To what extent does the search performance decreases for deeper and broader indexing schemas?
3. What is the benefit in precision and recall from querying indexes with more complex schemas?

Since indexing can be performed in the background and offline, we are only interested in drastic degradation for (1). Schema complexity appears not to affect search performance significantly. We see in Chart 1 that schema complexity only moderately affects indexing performance. With the most complex schema all documents are indexed in under 5 minutes. The estimation issues (2) and (3) are the important ones to be evaluated for a search application. Precision and recall mostly depend on the preciseness and quality of the query, availability of metadata in XML documents, the metadata indexing and querying strategy. The results were more than satisfactory addressing issue (2). For all queries for each index, we obtained results in under 0.5 seconds. We did not evaluate the system on a standard dataset. On the basis of the experiment with \(I_1, I_2\) and \(I_3\) we concluded that precision and recall increase for more complex schemas (3). The precision when querying \(I_3\) is higher than the one for \(I_1\), because it retrieves conceptually relevant documents either in English or Italian. Although many of the text terms are not annotated with concepts and the variety of queries is limited, the recall for \(I_3\) is comparable with the one from \(I_1\). The retrieval from \(I_2\) is with the highest precision and recall, since it combines advantages of \(I_1\) and \(I_3\).

5. Conclusion

The aim of this work was to define a user guided approach to XML retrieval that could be used for indexing and search in XML document corpora independent of document structure. We implemented our ideas in a platform independent search engine framework that combines structured and unstructured retrieval techniques and can be integrated in different kind of applications. We ended up with a middle layer component which so far is integrated into prototype systems created for the LT4eL [16] and AsIsKnown [15] research projects. Future work includes running experiments with the test collections created for the INEX competition. Our future goals include integration of automatic language detection and format conversion to XML. We also intend to implement and integrate different tokenizers and analyzers. A future aim and big challenge for us is to adapt and integrate the framework in a web search engine.

Acknowledgements

This work was supported by the Institute for Parallel Processing (Bulgarian Academy of Science) and granted by the European Projects AsIsKnown (FP6-028044) and LTILL (FP7-212578). I thank to my advisor Kiril Simov who laid the foundation of the work and assisted me during the past year and a half in the process of design, analysis, development and integration of the XML-IR framework.

6. References

[1] N. Fuhr: XML Information Retrieval and Information Extraction, University of Dortmund, Germany, 2003
[2] Ch. D. Manning, P. Raghavan, H. Schütze: Introduction to Information Retrieval, ISBN: 0521865719, Cambridge University Press, 2008.
[3] D. Carmel, N. Efriat, G. M. Landsau, Y. S. Maarek, and Y. Mass. An Extension of the Vector Space Model for Querying XML Documents via XML Fragments. Proceedings of the SIGIR 2002 Workshop on XML and Information Retrieval, 15. August 2002
[4] S. Amer-Yahia, M. Lalmas: XML search: languages, INEX and scoring. SIGMOD Record 35(4): 16-23 (2006)
[5] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram. XRank: Ranked Keyword Search over XML Documents. SIGMOD 2003.
[6] S. Cohen, J. Mamou, Y. Kanza, Y. Sagiv. XSEarch: A Semantic Search Engine for XML. VLDB 2003
[7] A. Theobald, G. Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. EDBT 2002
[8] A. Trotman, M. Lalmas. The Interpretation of CAS. INEX 2005
[9] E. Hatcher, O. Gospodnetić, and M. McCandless, “Lucene In Action”, ISBN: 1932394281, May 2008
[10] XQueryFull-Text: [http://www.w3.org/TR/xpath-full-text-10/](http://www.w3.org/TR/xpath-full-text-10/)
[11] Y. Mass, M. Mandelbrod. Retrieving the most relevant XML Components. INEX 2004
[12] INEX 2009: [http://www.inex.otago.ac.nz/](http://www.inex.otago.ac.nz/)
[13] N. Roussopoulos, S. Kelley, F. Vincent, Nearest Neighbor Queries, 1995
[14] N. Fuhr and G. Weikum. Classification and Intelligent Search on Information in XML. Bulletin of the IEEE Technical Committee on Data Engineering, 25(1), 2002.
[15] AsIsKnown: [http://www.asisknown.org/](http://www.asisknown.org/)
[16] LT4eL: [http://www.lt4el.eu/](http://www.lt4el.eu/)