PF/Tijah: text search in an XML database system

Djoerd Hiemstra, Henning Rode, Roel van Os and Jan Flokstra
University of Twente
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE Enschede
The Netherlands
{d.hiemstra, h.rode, r.vanos, j.flokstra}@cs.utwente.nl

ABSTRACT

This paper introduces the PF/Tijah system, a text search system that is integrated with an XML/XQuery database management system. We present examples of its use, we explain some of the system internals, and discuss plans for future work. PF/Tijah is part of the open source release of MonetDB/XQuery.

Keywords: Information Retrieval, XQuery, XML Databases, Open Source Systems

1. INTRODUCTION

PF/Tijah is a research project run by the University of Twente with the goal to create a flexible environment for setting up search systems by integrating the PathFinder (PF) XQuery system [3] with the Tijah XML information retrieval system [11]. The PF/Tijah system is part of the open source release of MonetDB/XQuery developed in cooperation with CWI Amsterdam and the University of München. The system is available from SourceForge.[1]

PF/Tijah will include out-of-the-box solutions for common tasks like index creation, stemming, result ranking (supporting several retrieval models), and relevance feedback, but it remains the same time open to any adaptation or extension. On the one hand, the system aims to be a general purpose tool for developing information retrieval (IR) end-user applications using XQuery statements with text search extensions. On the other hand, the system aims to be a playground for the information retrieval scientist and advanced user to easily set up and test new search systems. Advanced users can hook in the system at an intermediate level that provides the database scripting language MIL [4] and several pre-defined operations on terms and XML elements called Score Region Algebra operators [12].

[1] http://sourceforge.net/projects/monetdb/
http://monetdb-xquery.org

Second International Workshop on Open Source Information Retrieval (OSIR), 10 August 2006, Seattle, USA.

The PF/Tijah system has a number unique selling points that distinguish it from most other open source information retrieval systems.

- PF/Tijah supports retrieving arbitrary parts of the textual data, unlike traditional information retrieval systems for which the notion of a *document* needs to be defined up front by the application developer. For instance, if the data consist of scientific journals one can query for complete journals, journal issues, single articles, sections from articles or paragraphs with no need to adapt the index or any other part of the system configuration;
- PF/Tijah supports complex scoring and ranking of the retrieved results by means of so-called NEXI queries. NEXI [15] stands for *Narrowed Extended XPath*: a query language similar to XPath that only supports the descendant and the self axis step, but that is extended with a special `about()` function that takes a sequence of nodes and ranks those by their estimated probability of relevance to the query;
- PF/Tijah supports ad hoc result presentation by means of its query language. For instance, when searching for a special issue of a journal, it is easy to print any information from that retrieval result on the screen in a declarative way (i.e., not by means of a general purpose programming language), such as the special issue title, its date, the editors and the preface. This is simply done by means of XQuery element construction;
- PF/Tijah supports text search combined with traditional database querying, including for instance joins on values. For instance, one could search for employees from the financial department that also worked for the sales department and that sent an email about "tax refunds" (for an example, see below).

This paper is organised as follows. Section 2 presents several usage examples. Section 3 discusses practical design decisions and open problems. Section 4 concludes the paper.

2. SIMPLE USAGE EXAMPLES

PF/Tijah comes with several functions for indexing and retrieving XML data and for managing collections of XML

documents. In this paper, we will focus on the search functionality. We present two usage scenarios: Performing IR-only queries, and performing combined IR and database queries.

2.1 Performing IR-only queries

In order to understand how to use PF/Tijah, it is necessary to have some basic knowledge about using XQuery as for instance described by Katz et al. [9]. XQuery is a database query language for XML data that provides similar functionality as SQL does for relational data. XQuery is built around XPath, a simple path query language. In XPath, a path query such as `/html/head/title` selects from the root of an XML document first all elements tagged as `<html>`, then inside those all tagged as `<head>`, and finally inside those elements all `<title>` elements. PF/Tijah extends XQuery with a small number of user-defined functions, mainly the function `tijah-query()` that takes a sequence of nodes and a NEXI query and produces a sequence of nodes that is ranked by the estimated relevance to the query. NEXI [15] is a path language that supports an additional `about()` function that performs information retrieval queries on XML elements.

At several points in this paper, we will compare PF/Tijah to the draft XQuery full-text search standard that is currently under development by the World Wide Web Consortium [1]. PF/Tijah fulfills many goals of that standard and we might support parts of XQuery full-text in the future. Instead of putting all text search functionality inside user-defined functions as done by PF/Tijah, XQuery full-text extends XQuery with new language constructs for text search. This allows for a more natural way of embedding text search into XQuery. However, the XQuery full-text draft has problems as well: As shown below, XQuery full-text makes the end-user partly responsible for result scoring and ranking.

We will not explain XQuery, XQuery full-text, XPath and NEXI any further in this paper; instead we show some of the functionality of PF/Tijah by example queries, for instance:

```
let $c := doc("mydata.xml")
for $res in tijah-query($c, "//html[about(., ir db)];")
return $res/head/title
```

This query searches for all XML elements tagged as `<html>` using the text query "ir db". The query returns a ranked list of *titles* from those elements.[2]

The following example is slightly more complicated, and involves searching in two different parts of the data. Suppose we are interested in the following: *Give me paragraphs about XQuery in web pages about IR and DB.* This might be expressed as:

```
let $c := doc("mydata.xml")
for $res in tijah-query($c,
  "//html[about(., ir db)]//p[about(., xquery)];")
return $res
```

The query returns a ranked list of paragraphs. The top ranked paragraphs are most likely about "XQuery" and they are contained by web pages that are most likely about "IR" and "DB". So, the ranking is based on a final score that combines scores from the information retrieval query on `<html>` elements with the query on `<p>` elements. PF/Tijah combines scores of multiple `about()` functions in a – for the end user – transparant way, that is, the end user does not have to worry about scoring, score combination, and ranking. Using the W3C XQuery full-text standard, the query above might be expressed as:

```
(:  XQuery full-text instead of PFTijah :)
let $c := doc("mydata.xml")
for $res score $s in $c//p[. ftcontains "xquery" and
  ./ancestor::html ftcontains "ir" && "db"]
order by $s
return $res
```

Here, the user needs to explicitly tell the system to score the query results (by `score $s`), and the user needs to explicitly tell the system to rank the results (by `order by $s`).

By embedding NEXI into XQuery, PF/Tijah follows a different philosophy. In our opinion, result ranking is the *single most important requirement* of a search extension. Therefore, the default behaviour of a PF/Tijah text search query is to return a ranked sequence of nodes. Of course, the default behaviour might still be changed by using the `order by` clause, for instance if the user wants to order the results by their date of publication. In XQuery full-text, the default behaviour seems to be to return all matching nodes in document order, which is probably suboptimal in most applications.

An interesting question is how to combine the scores resulting from both `about()` functions in the PF/Tijah example query above. There are several acceptable ways to do so. Experiments have shown that the best methods for combining and propagating scores depend on the retrieval model that is used to calculate the scores in the first place [13]. PF/Tijah's default behaviour uses so-called language modeling scoring and the best performing combination and propagation methods for this model based on [13]. Advanced users can configure PF/Tijah such they can change the default behaviour of scoring; See Section 3.4. However, there are several problems with PF/Tijah's NEXI extension as well. Section 3.5 will address these further.

Currently, the old Tijah system is used mainly by ourselves, i.e., researchers that participate in scientific search evaluations organised by for instance the Text Retrieval Conference (TREC) [18] and the Initiative for the Evaluation of XML retrieval (INEX) [10]. We are aiming to support the same functionality with PF/Tijah to test for instance the effectiveness of information retrieval models, such as language modeling approaches. As an example, consider the evaluation task organised in the TREC video workshops [17], that provide a collection of videos and MPEG-7 XML annotations on those videos. The MPEG-7 annotations include text annotations from large-vocabulary speech recognition on the audio and optical character recognition on the video. The following query, taken from TRECVid topic 149, returns the identifiers of the top 1000 shots (tagged

[2]For historical reasons, the current implementation requires NEXI queries to end with a semi-colon

as 'videosegment') that contain Condoleeza Rice and their scores.

```
<videoSearchTopicResult tNum="0149"> {
  let $c := doc("trecvid.xml")
  let $q := "//VideoSegment[about(.//TextAnnotation,
    condoleeza rice)];"
  let $result := tijah-query-id($c, $q)
  for $node at $rank in tijah-nodes($result)
  where $rank <= 1000
  return <item seqNum="{$rank}" shotId="{$node/@id}"
    score="{tijah-score($result, $node)}" /> }
</videoSearchTopicResult>
```

In the example above, the returned item elements contain an attribute *score*.[3] Returning the scores of each node in a node sequence is a challenging problem when integrating XQuery with text search functionality. The XQuery data model supports sequences of simple types, but no sequences of complex types such as a sequence of sequences or a sequence of node/score pairs. In order to make scores available to the user, we introduce two functions: First, `tijah-queryid()` which executes the query and returns a result container (actually, the container is just an identifier, hence the name `tijah-queryid()`) that contains a sequence of scored nodes. Second, `tijah-nodes()` which takes the result container and returns a ranked sequence of nodes. So, the statement `tijah-nodes(tijah-query-id())` is equal the function `tijah-query()` introduced previously. The container, however, gives access to the scores of nodes as well by means of the function `tijah-score()`, which gives access to the scores of each retrieved element by returning the score of a node (or by returning a sequence of scores for a node sequence).

The use of separate score and node sequences is advocated by the XQuery full-text standard as well. The following expression of a `for` clause that contains an XQuery full-text `score` variable provides a nice integration of node selection and node scoring:

```
(:  XQuery full-text instead of PFTijah :)
for $result score $score in Expr
...
```

Currently, our system does not support any special language constructs, i.e., our queries are pure XQuery with user-defined functions. We might however support XQuery full-text syntax by evaluating the above as though it is replaced by the following expression, where `$id` and `$score` are new variables not appearing elsewhere:

```
let $id := tijah-query-id(Expr)
for $result in tijah-nodes($id)
let $score := tijah-score($id, $result)
...
```

We believe most users will never actually need these constructs, because the actual scores are often unimportant: It is the *ranking* of results by their score that is important. We like to stress again that PF/Tijah's text search extensions return ranked results; The user does not have to rank the results explicitly using `order by`, so there is often no need to have the actual scores of the retrieved nodes.

The XQuery-fulltext standard [1, Section 4.3.2] suggests another interpretation of their syntax for returning scores. They suggest to evaluate the expression as though it is replaced by:

```
(:  XQuery full-text instead of PFTijah :)
let $scoreSeq := fts:scoreSequence(Expr)
for $result at $rank in Expr
let $score := $scoreSeq[$rank]
...
```

Here as well, `$scoreSeq` and `$rank` are new variables, not appearing elsewhere. In this case, the query without the W3C language extension of the `for` clause results in a rather awkward XQuery statement which includes the text search query "`Expr`" twice.

2.2 Performing combined IR and XML database queries

PF/Tijah makes it possible to process queries that combine the results of an IR part (e.g. a text search query expressed in NEXI) and an DB part (expressed in XQuery). A simple example query for this case might perform a restrictive database selection, for instance give me all documents written by a certain author, and do an IR query only on the publications of that author, so *From a database with research papers, give me titles of documents that the author Vojkan Mihajlovic wrote about "open source XML retrieval".*, which might be expressed as:

```
let $vm := doc("source1.xml")/DOC[author =
  "Vojkan Mihajlovic"]
for $res in tijah-query( $vm,
  "//text[about(.,open source xml retrieval)];")
return $res/title/text()
```

Similar functionality is provided by many other search engines, i.e., this will rank papers about "open source XML retrieval" on top, but the ranking includes only the papers by Vojkan Mihajlovic.

More advanced examples might use joins. A join combines information from two or more items in the database by putting together those items that share the same value for a data item. Query processing in relational databases heavily depends on joining tables on the values of columns. An example of a query that probably cannot be processed by most of todays search engines is the following: *From Shakespeare's plays, give me all speakers from the second part of Henry the Sixth, that speak about a "bloody murder", and that also spoke in first part of Henry the Sixth.* This might be formulated as:

```
for $doc in tijah-query(doc("hen_vi_2.xml"),
  "//SPEECH[about(.,bloody murder)];")
where $doc//SPEAKER = doc("hen_vi_1.xml")//SPEAKER
return $doc//SPEAKER
```

This query ranks the requested speakers by the probabil-

[3]For TRECVid, the score of an item does not need to be returned, but we included it here for illustrative purposes.

ity that they in fact talked about a bloody murder. Such a query might seem a bit far-fetched for searching Shakespeare's plays, but similar queries would be helpful in enterprise search scenarios. For instance, suppose the chief executive officer of a large multinational looks for people that have experience at both the financial department and the sales department and are an expert on "sales tax refunds". He or she might search the enterprise data for the following: *From our enterprise database, give me all employees from the financial department, that sent an email about "sales tax refunds" and that also worked for the sales department.* Obviously, this would be formulated in a similar way as the Shakespeare query above.

As another example, consider the following information need taken from INEX topic 14 [10]: *Find figures that describe the Corba architecture and the paragraphs that refer to those figures. Retrieved components should contain both the figure and the paragraph referring to it.* This might for instance be expressed as:

```
let $doc := doc("inex.xml")
for $p in tijah-query($doc,
  "//p[about(.,corba architecture)];")
for $fig in $p/ancestor::article//fig
where $fig/@id = $p//ref/@rid
return <result> { $fig, $p } </result>
```

Interestingly, the figures and the paragraphs that refer to them might be relatively far apart in the XML data. This query will find figures that do not mention "corba" and "architecture" in their caption (for instance the caption might contain "Object management architectural overview"), but that do mention "corba architecture" in the paragraph when referring to the figure. INEX topic 14 was released in 2002, but was at that time considered to be too difficult by the organisers, or at least, topic 14 at that time needed functionality that none of the participating systems implemented. Another problem with such queries is that they might be hard to assess, that is, it might be hard to determine what elements are good answers. Although there has been an INEX conference each year since 2002, INEX has not taken up the challenge of join queries until today.

3. DESIGN ISSUES AND OPEN PROBLEMS

Started as completely independent academic research projects, Pathfinder and Tijah followed quite different goals – the first one to become an efficient XQuery compiler, the second to work as an XML IR engine. However, both systems share similar internal data models of the XML content and both have been implemented and tested mainly to work on the MonetDB backend. PF/Tijah is a module of Pathfinder that is based on the old Tijah system and developed by we following six general design constraints in order of their importance:

1. Keep Pathfinder untouched by our extension;
2. Use existing Pathfinder functionality as much as possible;
3. Support generalized retrieval methods instead of specialized (document) retrieval;
4. Enable fast retrieval;
5. Minimize redundant storage of data;
6. Use existing Tijah functionality as much as possible.

3.1 Putting things together

PF/Tijah is compiled as a module in Pathfinder. We are using the following elements from Pathfinder:

The document shredder: when loading documents into collections, these documents are shredded by the Pathfinder shredder. This way we inherit all of Pathfinder's document loading properties: speed, loading from any URL, caching, DTD processing, etc.

The XML serializer: when documents have been shredded by Pathfinder, they must be indexed by PF/Tijah. We have constructed a new driver for the existing document serializer in Pathfinder that creates the PF/Tijah index.

The loop-lifted descendant step implementation: instead of using Tijah's containment join implementation we re-use the Pathfinder descendant-step implementation, to minimize the size of the PF/Tijah module. Furthermore, PathFinder staircase joins might be faster than the old Tijah containment joins as they have been well-studied [5, 7].

We are using the following elements from Tijah:

The NEXI query processor: Tijah uses the NEXI query language to express structured information retrieval queries. The module translates NEXI queries into SRA expressions.

Score Region Algebra (SRA): SRA is an algebra for expressing structured information retrieval queries at a logical level [12], i.e, between the query language (conceptual layer) and the database engine and index (physical layer). SRA expressions are independent of the database back-end or storage scheme that is used.

Pathfinder and the old Tijah system use a similar datamodel. Pathfinder uses a pre-/post-order encoding of the XML documents [3], whereas Tijah uses a so-called region data model, i.e. a start-end encoding of the XML document [6], where the region starts are in pre-order and the region endings in post-order. However, PF/Tijah cannot directly run on the Pathfinder index. The main difference is the need to index words instead of nodes. Every occuring word has to be assigned to a new pre-order identifier, not like in Pathfinder where a whole text node is regarded as one unit. PF/Tijah therefore splits text strings into word tokens while keeping their order with the pre-order numbering. This allows to perform simple keyword queries as well as phrase and proximity search.

For integrating the 2 systems, the following points took the most consideration:

PF-light index: The index structure of PF/Tijah is a light version of the Pathfinder index. It is created next to the existing Pathfinder index, which is still used for the normal XQuery evaluation. Apart from indexing single words here instead of whole text nodes, it knows additional inverted structures sorted on term identifiers

to accelerate the selection of term occurrences in the collection. On the other hand, it is still a *light* index in the sense that it doesn't replicate the full document content. Attributes, Processing Instructions, etc. are kept in the normal Pathfinder index only, since they will not be addressed with NEXI queries.

A further problem arising, working with 2 different indices, is the translation between both. A sequence of nodes created by an arbitrary XQuery expression that is further used in a search process, has to be translated from Pathfinder pre-order identifiers to their corresponding ones in the PF-light index. The same vice versa for passing the ranked result sequence back and assigning them to an XQuery variable.

SRA operator implementations: All SRA operators needed a new implementation on the physical layer now working on the new PF-light index. Special attention was given here to enable fast term selection and to employ the highly performance trimmed containment-join operators coming with the Pathfinder system.

3.2 The MonetDB backend

In contrast to many other open source IR environments like Lucene [8], Lemur/Indri [14], or Terrier [16] our system is set up on a light database backend, the MonetDB system. Whereas pure IR systems can profit from highly specialized algorithms, PF/Tijah gains a lot flexibility by abstraction from the physical layer. Search extensions are easily programmed as database scripts, pre- and post-processing of the data can be done without the need to extend the system. Furthermore, since database operations are highly tuned to efficient data processing, many IR operations can keep up with, or even beat, the efficiency of specialized IR systems.

The used database backend, MonetDB [4], is an open source main memory database system. It is a light-weight system compared to common commercial products, but allows highly effective data processing. If required, new database operations can be added to the database kernel. The main limitation of the system is its restriction to process queries completely in main memory. Our IR tools manage collections up to several gigabytes on a single machine, currently. This limitation will be overcome by a new version of the database backend [19].

3.3 Collection handling

PF/Tijah comes with additional functionality for handling collections of a large number of smaller XML documents. This functionality includes naming a collection (or database) adding a document to the collection, and *finalizing* the collection, that is, building the inverted files. Additionally, PF/Tijah can be configured in such a way that the XML data is not loaded in the Pathfinder system at index time. In this way, only PF/Tijah's Pathfinder light index is built. At query time, if for instance information has to be returned of a top 10 retrieved by a NEXI query, Pathfinder loads only those XML documents that are actually returned by the query.

3.4 System configuration

The functions `tijah-query()` and `tijah-query-id()` allow the inclusion of any number of options by means of a special empty XML element `<TijahOptions/>` as follows.

```
let $opt := <TijahOptions algebraType="COARSE2"
  txtmodel_model="NLLR" />
let $c := doc("mydata.xml")
for $res in tijah-query($opt, $c, "//html[about(., xml)];")
return $res//title
```

With these options, the user can overwrite the default behaviour for query plan generation (using the `algebraType` attribute), for scoring in an about clause (using the attribute `txtmodel_model`), but also for score combination and propagation, etc.

3.5 Problematic effects of the NEXI language embedding

Currently, NEXI queries are embedded as strings in XQuery expressions. The NEXI query string remains a black box for XQuery, so no variable usage or static type checking and syntax checking is possible in XQuery. A tighter integration is desirable, for instance as recommended by the XQuery full-text standard.

Another problem is that NEXI and XQuery share some expressiveness. Both formalisms are able to process simple path queries. For example, the following queries are currently equivalent:

```
let $c := doc("test.xml")
return tijah-query($c//DOC//TEXT, "//P[about(.,XML)];")
```

and

```
let $c := doc("test.xml")
return tijah-query($c, "//DOC//TEXT//P[about(.,XML)];")
```

In future versions of the system, the text search functions could be less expressive. This however, might introduce problems with scoring complex queries as shown in Section 2.1.

4. CONCLUSIONS

PF/Tijah aims at supporting search applications for highly structured, heterogenous content on medium-sized XML collections. The system can be used to quickly develop retrieval applications. It is part of the open source release of MonetDB/XQuery (version 0.13.1 and higher) and available from SourceForge.

To develop PF/Tijah, we made choices that are currently not recommended by the XQuery full-text draft standard. Our choices are driven by a focus on *ranking* search results. In our opinion, result ranking is the *single most important* requirement for text search. If the ranking procedures are not effective, advanced features like implicit sentences and paragraphs, thesaurus queries, queries across elements, etc. will be of limited use. Currently, XQuery full-text only gives limited thought to scoring. In the W3C use cases draft [2] only one of 16 use case sections is devoted to result ranking. The other sections assume that elements of text search queries are returned in document order.

Note that we do not advocate to define scoring in any way by the standard: On the contrary, scoring must be implementation-defined, and users should not be bothered by explicit

ranking of the results using XQuery `order by` and not be bothered by explicit combination of scores: The `tijah-query()` function of the PF/Tijah module returns a *ranked* sequence of nodes, and the NEXI query language supports simple axis steps to enable powerful score combination.

For research at the University of Twente, PF/Tijah serves as our "Swiss knife" for research into information retrieval and research into the integration of information retrieval and databases. With PF/Tijah, it is relatively easy to experiment with new retrieval approaches, and it is relatively easy to test the implications and limitations of approaches to XML information retrieval: those advocated by XQuery full-text as well as those advocated by for instance the INEX workshops.

More information on PF/Tijah is currently available from the project wiki at: http://monetdb.cwi.nl/projects/trecvid/MN5/index.php/PFTijah_Wiki

Acknowledgements

This research is funded by the Dutch BSIK progamme MultimediaN. Many thanks to Vojkan Mihajlović, Thijs Westerveld (CWI Amsterdam), Georgina Ramírez (CWI Amsterdam), and Arjen de Vries (CWI Amsterdam) for helpful advice and for testing the system.

5. REFERENCES

[1] S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, M. Holstege, D. McBeath, M. Rys, and J. Shanmugasundaram. XQuery 1.0 and XPath 2.0 full-text. Technical report, Word Wide Web Consortium. working draft 1 May 2006, http://www.w3.org/TR/xquery-full-text

[2] S. Amer-Yahia and P. Case. XQuery 1.0 and XPath 2.0 full-text use cases. Technical report, Word Wide Web Consortium. working draft 1 May 2006, http://www.w3.org/TR/xmlquery-full-text-use-cases/

[3] P.A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and Jens Teubner. MonetDB/XQuery: A fast XQuery processor powered by a relational engine. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2006.

[4] P.A. Boncz. *Monet: A Next-Generation DBMS Kernel for Query-Intensive Applications*. PhD thesis, University of Amsterdam, 2002.

[5] P.A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. Loop-lifted staircase join: from XPath to XQuery. Technical Report INS-E0510. *CWI, Amsterdam*, 2005.

[6] F.J. Burkowski. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Texts. In *Proceedings of the 15th ACM SIGIR Conference on Research and Development in Information Retrieval*, 1992.

[7] T. Grust, M. van Keulen, and J. Teubner. Staircase join: Teach a relational DBMS to watch its (axis) steps. In *Proceedings of the 29th Conference on Very Large Databases (VLDB)*, 2003.

[8] E. Hatcher and O. Gospodnetic. *Lucene in action*. Manning Publications, 2005.

[9] H. Katz, D. Chamberlin, D. Draper, M. Fernandez, M. Kay, J. Robie, M. Rys, J. Simeon, J. Tivy, and P. Wadler. *XQuery from the Experts: A Guide to the W3C XML Query Language*. Addison Wesley, 2003.

[10] M. Lalmas and G. Kazai. Report on the ad-hoc track of the INEX 2005 workshop. *SIGIR Forum* 40(1):49–57, 2005.

[11] J. List, V. Mihajlovic, G. Ramirez, A.P. de Vries, D. Hiemstra, and H.E. Blok. Tijah: Embracing information retrieval methods in XML databases. *Information Retrieval Journal* 8(4):547–570, 2005.

[12] V. Mihajlovic. Score region algebra: A framework for structured information retrieval. In *SIGIR Doctoral Consortium Workshop*, 2005.

[13] V. Mihajlovic, H.E. Blok, D. Hiemstra, and P.M.G. Apers, Score Region Algebra: Building a Transparent XML-IR Database. In *Proceedings of the 14th International Conference on Information and Knowledge Management (CIKM)*, 2005

[14] P. Ogilvie and J. Callan. Experiments using the Lemur toolkit. In *Proceedings of the tenth Text Retrieval Conference (TREC)*, 2001.

[15] R. A. O'Keefe and A. Trotman. The simplest query language that could possibly work. In *Proceedings of the 2nd Initiative for the Evaluation of XML Retrieval (INEX)*. ERCIM workshop proceedings, 2004. http://www.cs.otago.ac.nz/postgrads/andrew/2003-4.pdf.

[16] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and D. Johnson. Terrier information retrieval platform. In *Proceedings of the 27th European Conference on Information Retrieval (ECIR)*, 2005.

[17] A.F. Smeaton, P. Over, and W. Kraaij. TRECvid: evaluating the effectiveness of information retrieval tasks on digital video. In *Proceedings of ACM Multimedia*, pages 652–655, 2004.

[18] E.M. Voorhees and D. Harman, editors. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005.

[19] M. Zukowski, S. Heman, A.P. de Vries, and P. Boncz. Efficient and Flexible Information Retrieval Using a Relational Database Engine. *submitted*, 2006.