

# Applying the IRStream Retrieval Engine to INEX 2003

Andreas Henrich, Volker Lüdecke  
University of Bamberg  
D-96045 Bamberg, Germany  
{andreas.henrich|volker.luedecke}@wiai.uni-bamberg.de

Günter Robbert  
University of Bayreuth  
D-95440 Bayreuth, Germany  
guenter.robber@uni-bayreuth.de

## ABSTRACT

Last year, in the context of the INEX evaluation initiative, we could show that our retrieval system IRStream is successfully applicable as a retrieval engine for XML-documents. Nevertheless, we have to point out that IRStream can be further optimized in many directions.

In the present paper we show, how IRStream was extended and improved for its application to INEX 2003 in order to achieve better retrieval results. Furthermore, we present some first retrieval results, which demonstrate the impact of the improvements of IRStream concerning the quality of the retrieval result.

## 1. MOTIVATION

Last year, as a participating organization at the INEX evaluation initiative [11], we applied IRStream to the collection of XML documents provided by INEX. Hereby, we investigated the usability of IRStream for structured text documents. By the application of IRStream as retrieval system for XML-documents, we have recognized that IRStream can be further improved and optimized in many respects.

As two of the main drawbacks of IRStream we have identified the absence of a component for an automatic generation of queries based on topic data and the problem that IRStream sometimes provided wrong granules as the result of a query. Therefore we decided to improve and extend IRStream in order to avoid the drawbacks mentioned above.

IRStream in this respect is intended to provide a powerful framework to search for components of arbitrary granularity – ranging from single media objects to complete documents. IRStream combines traditional text retrieval techniques with content-based retrieval for other media types and fact retrieval on meta data. In contrast to other retrieval services which permit set-oriented or navigation-oriented access to the documents, we argue for a *stream-oriented* approach. In the following paper, we shortly describe the significant features of this approach and point out the system architecture of IRStream. Furthermore, we present the application of an extended and improved version of our IRStream retrieval engine as a retrieval system for XML documents in the context of INEX 2003 [4].

The rest of the paper is organized as follows: In section 2 we will give a short overview of the ideas and main components

of IRStream. The concrete architecture of our IRStream implementation is presented in section 3. Section 4 shows how we improved our retrieval system IRStream in order to use it as a retrieval engine for XML documents in the context of INEX 2003. Afterwards in section 5, we present some first experimental results concerning the improved version of IRStream. Section 6 concludes the paper.

## 2. STREAM-ORIENTED QUERY PROCESSING

“Stream-oriented” means that the entire query evaluation process is based on components producing streams one object after the other. First, there are components creating streams given a base set of objects and a ranking criterion. We call these components *rankers*. Other components consume one or more input streams and produce one (or more) output stream(s). *Combiners*, *transferers* and *filters* are different types of such components.

### 2.1 Rankers

The starting point for the stream-oriented query evaluation process are streams generated for a set of objects based on a given ranking criterion. For example, text objects can be ranked according to their content similarity compared to a given query text and images can be ranked with respect to their color or texture similarity compared to a given sample image.

Such “initial” streams can be efficiently implemented by access structures such as the M-tree, the X-tree, the LSD<sup>h</sup>-tree, or by approaches based on inverted files. All these access structures can perform a similarity search in the following way: (1) the similarity search is initialized and (2) the objects are taken from the access structure by means of some sort of “getNext” method. Hence, the produced streams can be efficiently consumed one element after the other.

### 2.2 Combiners

Components of this type combine multiple streams providing the same objects ranked with respect to different ranking criteria. Images are an example of media types, for which no single comprehensive similarity criterion exists. Instead, different criteria addressing color, texture and also shape similarity are applicable. Hence, components are needed which merge multiple streams representing different rankings of the same base set of objects into a combined ranking.

Since each element of each input stream is associated with some type of retrieval status value (RSV), a weighted average of the retrieval status values in the input streams can be used to derive the overall ranking [3]. Other approaches are based on the ranks of the objects with respect to the single criteria [12, 7]. To calculate such a combined ranking efficient algorithms, such as Fagin’s algorithm [1, 2], Nosferatu [14], Quick Combine [5] and  $J^*$  [13] can be deployed.

### 2.3 Transferers

With structured documents, ranking criteria are sometimes not defined for the required objects themselves but for their components or other related objects. For example, searching for images where the text in the “vicinity” (for example in the same section) should be similar to a given sample text. In such situations the ranking defined for the related objects has to be transferred to the desired result objects.

To put it more precisely, we are concerned with a query which requires a ranking of objects of some desired object type  $ot_d$  (image for example). However, the ranking is not defined for the objects of type  $ot_d$ , but for related objects of type  $ot_r$  (text for example).

We assume that the relationship between these objects is well-defined and can be traversed in both directions. This means that we can determine the concerned object - or objects - of type  $ot_d$  for an object of type  $ot_r$  and that we can determine the related objects of type  $ot_r$  for an object of type  $ot_d$ . The concrete characteristics of these traversal operations depend on the database or object store used to maintain the documents. In objectrelational databases join indices and index structures for nested tables are used to speed up the traversal of such relationships. For a further improvement additional path index structures can be maintained on top of the ORDBMS (cf. section 3).

Furthermore, we assume there is an input stream yielding a ranking for the objects of type  $ot_r$ . For example, this stream can be the output of a ranker or combiner.

To perform the actual transfer of the ranking we make use of the fact that each object of type  $ot_r$  is associated with some type of retrieval status value ( $RSV_r$ ) determining the ranking of these objects. As a consequence, we can transfer the ranking to the objects of type  $ot_d$  based on these retrieval status values. For example, we can associate the maximum retrieval status value of a related object of type  $ot_r$  with each object of type  $ot_d$ . Another possibility would be to use the average retrieval status value of all associated objects of type  $ot_r$ . In [10] you will find a detailed description of an algorithm called “RSV-Transfer”, which is used by IRStream to perform the transfer of rankings between different object types.

### 2.4 Filters

Of course, it must be possible to define filter conditions for all types of objects. Accordingly, it is necessary that filter components are used for our stream-oriented approach. These filter components are initialized with an input stream and a filter condition. Then only those objects from the input stream which fulfill the given filter condition are passed to the output stream.

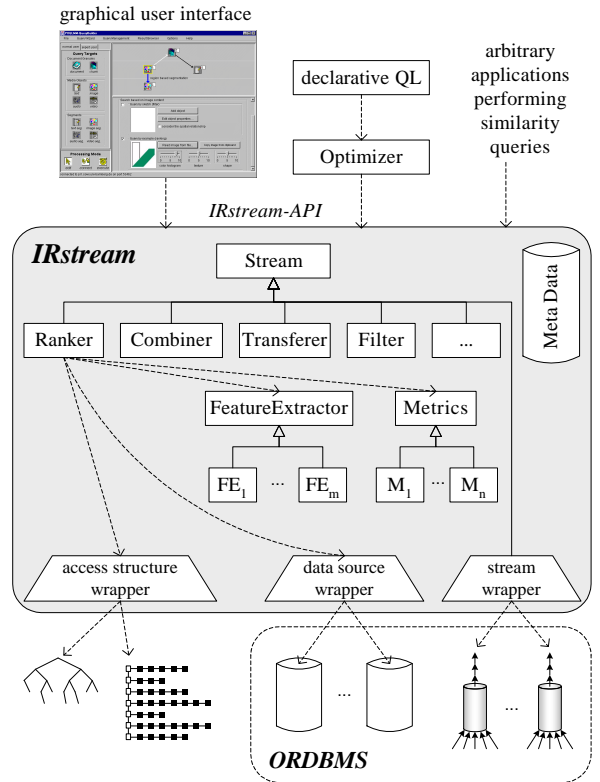


Figure 1: Architecture of the IRStream system

## 3. THE IRSTREAM ARCHITECTURE

The architecture of our IRStream system is based on the idea that the data is maintained in external data sources. In our implementation, an ORDBMS is used for this purpose. The stream-oriented retrieval engine is implemented in Java on top of this data source and provides an API to facilitate the realization of similarity based retrieval services. Figure 1 depicts this architecture.

The core IRStream system — shaded grey in figure 1 — comprises four main parts: (1) Implementations for rankers, combiners, transferers, and filters. (2) Implementations of various methods for the extraction of feature values as well as corresponding similarity measures. (3) A component maintaining meta data for the IRStream system itself and applications using IRStream. (4) Wrappers needed to integrate external data sources, access structures and stream implementations.

### Feature Extractors and Similarity Measures

A feature extractor receives an object of a given type and extracts a feature value for this object. The similarity measures are methods which receive two feature representations — usually one representing the query object and an object from the database. The result of such a similarity measure is a retrieval status value.

### Ranker, Combiner, Transferer, Filter, ...

All these components are subclasses of the class “Stream”.

The interface of these classes mainly consists of a specific constructor and a `getNext` method.

For example, the constructor of a *ranker* receives a specification of the data source, a feature extractor, a similarity measure and a query object. Then the constructor inspects the meta data to see if there is an access structure for this data source, this feature extractor, and this similarity measure. In this case, the access structure is employed to speed up the ranking. Otherwise, a table scan with a subsequent sorting is performed.

For the construction of a *combiner* two or more incoming streams with corresponding weights have to be defined. Here it is important to note that combiners such as Fagin's algorithm or Quick Combine rely on the assumption that random access is supported for the objects in the input streams. The reason for this requirement is simple. When these algorithms receive an object on one input stream, they want to calculate the mixed retrieval status value of this object immediately. To this end, they perform random accesses on the other input streams. Unfortunately, some input streams are not capable of such random access options, or a random access would require an unreasonable high effort. In these cases, other combine algorithms — such as Nosferatu or  $J^*$  — have to be applied.

For the construction of a *transferer*, an incoming stream, a path expression and a transfer semantics have to be defined. In our implementation, references and scoped references provided by the underlying ORDBMS are used to define the path expressions.

To construct a *filter*, an incoming stream and a filter predicate have to be defined.

## Meta Data

This component maintains data about the available feature extractors, similarity measures, access structures, and so forth. On the one hand, this meta data is needed for the IRStream system itself in order to decide if there is a suitable access structure for example. On the other hand, the meta data is also available via the IRStream-API for applications.

## Wrapper

IRStream makes the extension of the retrieval service in various directions possible by the use of wrappers and interfaces: *Data source wrappers* are needed to integrate systems maintaining the objects themselves into our retrieval system. At present, objectrelational databases can be used via JDBC. Whereas *access structure wrappers* can be used to deploy access structures originally not written for our system. For example, we incorporated an LSD<sup>h</sup>-tree written in C++ via a corresponding wrapper. In contrast, the *stream wrapper interface* is used to incorporate external sources for streams into our system. It can be used to incorporate external stream producers. At present, the text module of the underlying ORDBMS is integrated via a stream wrapper.

On top of the IRStream API various types of applications can be realized. An example is a graphical user interface where the user can define the query as a graph of related query objects [9]. Another possibility is to implement a declarative query language on top of the API. At present, we are working on a respective adaptation of our POQL<sup>MM</sup> query language [6, 8].

## 4. EXTENSIONS AND IMPROVEMENTS OF IRSTREAM FOR INEX2003

This section describes the main improvements applied to the IRStream retrieval system in the context of INEX 2003. For that, every retrieval system had to be able to perform an automatic query generation from topic data. While a topic is interpreted as a representation of an information desire, a query in this context is an internal representation for the system's retrieval process. Thus, the first extension of IRStream was to integrate a query generation step into this retrieval process. An evaluation of last year's results shows that one of main problems of IRStream02 was the determination of a fitting granule of retrieval results for CO-topics, and furthermore an automatic processing of structural constraints of CAS-topics, as well as automatically generating multiple results from one document (e.g. a list of authors). To solve these problems, the retrieval process of the system was completely redesigned, which is described in this section.

To determine fitting granules for retrieval results (and their corresponding identifying paths), a retrieval system has to be able to perform two tasks: First, to extract (possibly several) fragments of one document and to determine their unique paths (including node indices). In this case a path expression is given as part of the query, which describes a structural constraint for result granules, as is the case with CAS-topics. Second, the system must be able to process queries which do not contain a constraint regarding the result granule (CO-topics). In this case, the decision on the fitting granule is to be made automatically within the retrieval process.

### 4.1 Automatic query generation

The queries used internally by a retrieval system, generated from the topic data, may influence the quality of retrieval results significantly. In order to compare the results of different retrieval systems or even the result of a retrieval system in various development states, the influence of manual (pre-) processing must be eliminated. Therefore an automatic query generation was added to the IRStream system, which was also a requirement for retrieval systems participating in INEX 2003. For reasons of performance, two different approaches for CO- and CAS-topics were used, although every CO-topic may be converted into CAS-format by interpreting a CO-topic title as `//[about(., 'CO-title')]`. The different retrieval processes for these two topic types will be described later in this section.

The general architecture is the same for both variants. A wrapper-class *Topic* parses a topic file and provides means of access via a Java-API. The system is thereby also prepared for changing topic-formats, which will result in additional sub-classes of this wrapper. The methods provided

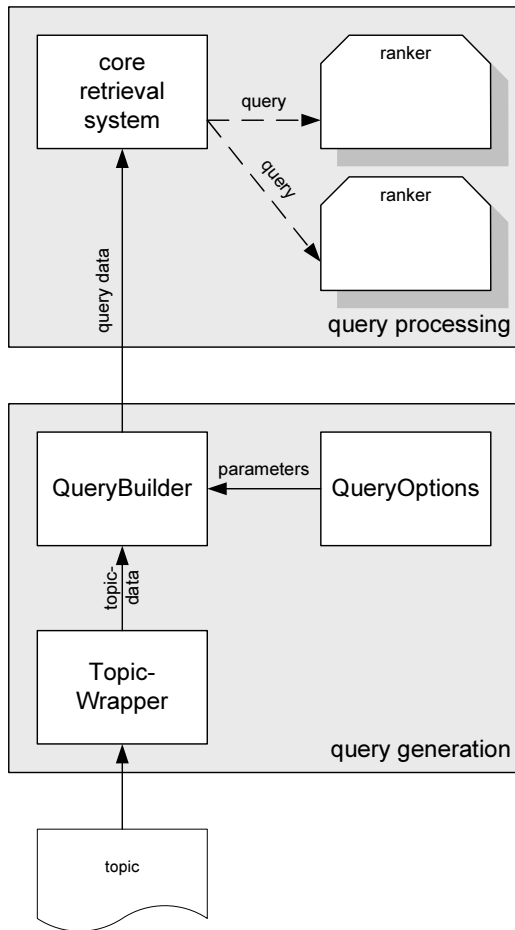


Figure 2: Architecture of query generation

by *Topic* are used by a *QueryBuilder* component specialized in CO-topics or CAS-topics respectively. This component creates the queries internally used by the *Rankers* of the core-retrieval system. To configure the query generation, a *QueryOptions* class is used, which contains all kinds of parameters used in the generation process. Figure 2 gives an overview of the general architecture and the differentiation between query generation and query processing.

Every query may make use of any of the following three topic parts: the *title*, the *description* and the *keywords*. Within the topic title, terms may further be categorized in must-terms (marked by a +), must-not terms (marked by a -) and terms not marked at all. For each part or each category of terms, the *QueryOptions* class contains parameters about:

*Consideration*: Shall these terms be considered for query generation at all?

*Weighting*: What weight shall be associated to these terms (1-10)?

*Stemming*: Shall the stemming operator of the underlying ORDBMS be used for these terms?

*Connectors*: Which connecting operator (OR, AND, AC-CUMulate) shall be used to connect terms of this class

or between classes of terms?

*Compound terms*: Which way shall compound terms be treated?

## 4.2 CAS-topics

A CAS-topic contains structural constraints as well as content information, so that three logic parts of a CAS-topic may be identified: First, a constraint regarding the granule of result elements. Second, content and structure information about the result element itself — i.e. its inner context —, which shall be called *content constraint*. Third, there may be content and structure information about the result element's parent or sibling elements — i.e. the element's environment —, which shall be called *structure constraint*.

The differentiation between content and structure constraint may easily be done by looking at the syntax of a CAS-topic title:

```
[node [filter]]* target-node filter
```

Every filter (which corresponds to constraints) before the target-node belongs to the structure constraint, while the filter given for the target-node contains the content constraint.

The title of a CAS-topic contains a path expression that must be matched by the path of a result element. For the automatic query generation, this path expression is simply the concatenation of all nodes. Normally, there are several elements within a document with matching paths, since the path expression may contain wildcards and does not have to use node indices. Thus, a retrieval system not only has to find relevant documents and determine fitting sub-elements of that element, but it also has to determine relevance scores for each sub-element. Therefore we inserted a new table into the underlying ORDBMS which contains every addressable element of the document collection, i.e. every element that matches the XPath-expression `//*`, which are about 8 mio elements. Each table entry consists of an element with all its sub-elements and their textual content, its unique path expression, and its path expression without indices. To determine the unique path of an element, which is needed for the creation of the submission-file, this data can simply be read from this table. To fulfill the structural constraint of a CAS-topic regarding the result granule, only a selection of those elements is evaluated whose path matches the path expression given in the topic title. Apparently, this approach implies a high degree of redundancy, since the table contains every textual content multiple times. Further developments of IRStream will address this problem, probably by making extended use of the transferer functionality.

The content constraint includes every information that is given about the result element itself. That may be content only, but also constraints concerning the internal structure of an element, like a section having a title about **information retrieval**:

```
/article/bdy/  
  sec[about(.,//st,"information retrieval")]
```

The crucial factor of this logic part of the topic is that every information needed is within the result element itself and thus may be addressed via the table mentioned above.

The structure constraint includes every information given about the environment of the result element, i.e. its sibling and parent elements. This may include both structure and content information which is not contained in the result element itself and therefore cannot be addressed via the table mentioned above, since the table entries are decoupled from their environment. To fulfill this constraint, a document as a whole has to be evaluated, i.e. it refers to a whole article instead of a result element only.

By looking at an example (topic 77), the retrieval process of IRStream for CAS-topics and the integration of a query generation step into this process will be depicted. The title of topic 77 states:

```
//article[about(./sec,'reverse engineering')]/
  sec[about(.,'legal') OR about(.,'legislation')]
```

The concatenation of all nodes is `//article//sec`, which is the given path expression that all result elements have to fulfill. Therefore only elements with the fitting granule will be ranked in the query process, which is implemented via a corresponding `WHERE`-clause.

The content constraint, referring to the result element itself, is contained in the last filter. It says that the result element has to be about concepts of `legal` or `legislation`. The query generation component successively reads all about-clauses and their connectors. Each about-clause is translated into a corresponding `INPATH`-clause of the ORDBMS, which reads (`terms INPATH path`) and includes any given structural constraints. In this example, (`legal INPATH /sec`) would be the resulting query part. The `INPATH`-clauses, their connectors and the result element's path expression form the main part of the content query, which is applied to the table containing every addressable element.

The structure query on the other hand has to be applied to a table of whole articles, which contain the complete structure information of a document. The query generation is done accordingly, reading each filter successively and connecting the resulting `INPATH`-expressions. The last filter in the topic-title may or may not be part of the structure query. Not including it means that some articles are probably marked relevant that do not contain any elements that satisfy the content constraint. IRStream therefore considers the content query to be a part of the structure query.

In order to get a result ranking, these two queries have each to be processed by a ranker-component and then be joined into a final ranking. These two rankers create streams of two different object types — article (structure query) and element (content query) —, which cannot directly be combined by a combiner-component. Therefore a transferer-component is needed, which transfers the ranking of an article to all its sub-elements. A special filter-component filters all elements whose path does not fulfill the given path expression. The output of this filter is a stream of elements,

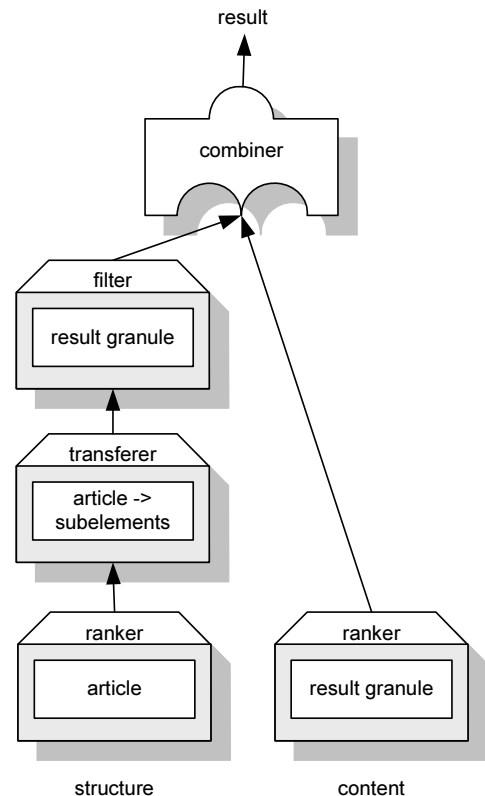


Figure 3: CAS-topic processing

and thus a combiner can finally merge the two streams into a result ranking. This procedure is shown in figure 3.

Obviously, this (general) procedure can be optimized, because the transferer creates hundreds of elements that are immediately eliminated by a filter. Therefore the task of ranking, transferring and filtering was integrated into a specialized component `InexRanker`, which relocates the transferring-process into the DBMS. The three logical steps described above can thereby be performed by a single SQL-query:

1. ranking an article in reference to the structure query
2. transferring the RSV to all sub-elements, identified via foreign key relationship
3. selection of those elements that fulfill the given path expression

### 4.3 CO-topics

The special challenge while processing CO-topics is that the retrieval system has to decide autonomously, which granule of the result elements is the most fitting. For INEX 2003, the procedure for handling CO-topics is based on the table mentioned above, which contains every addressable element including all its textual content and that of its sub-elements. A single ranker-component simply creates a ranking of all

those elements, and an element's filename and unique path may be read from this table. The aim of this approach was to evaluate whether it is worthwhile basing further optimizations on it, which are obviously possible, since this table contains about eight million elements, every layout-tag (italic, bold etc.) being contained.

For CO-topics, four characteristics can be identified. Based on these, the general applicability of this approach is to be shown:

*CO-topics do not contain structural information*

The elements in the table used are decoupled from their structural environment and are treated as single documents. No structure information is needed for this query processing.

*CO-topics do not contain constraints regarding the granule of result elements*

By this procedure, elements of all granules are ranked likewise, so that every granule may be contained in the result ranking. Possible optimizations will be addressed in section 6.

*An ideal result element satisfies the information need completely*

A retrieval system cannot validate a complete answering of an information need, but this requirement has to be considered in the process of determining relevance scores. Regarding an XML-document as being a tree of elements, that one element obviously fulfills that requirement best, which is superior to all elements which contain relevant information. If several paragraphs are marked as relevant, for example, their corresponding section seems to be the best fitting element. The calculation of a *score*-value that is done by the underlying ORDBMS provides an according evaluation, because it is in principle based on absolute term frequencies. Thus, superior elements normally get a relevance score which is equal to or greater than that of their child elements.

*An ideal result element is specific about the topic's theme*

For INEX 2003, IRStream did not eliminate multiple result elements within a branch of the document tree, the consequences of which with respect to retrieval effectiveness has not yet been evaluated, but it will be addressed in the near future. If several elements of a branch have the same RSV-score, it is obviously the smallest element that conforms best to this requirement. It remains to be seen whether elimination of such duplicates or considering document lengths will improve retrieval effectiveness.

The query generation for CO-topics is similar to that of CAS-topics, but here only one query has to be created, and no structural information has to be included. Terms in the title of CO-Topic may be marked by a + (declared as must-terms). The IRStream query generation allows to interpret these markings as strict or vague. A strict interpretation means that only those elements may be relevant that contain

all must-terms. Therefore these terms are connected to each other by AND-operators, and must-terms and all other terms are each encapsulated by brackets which are also connected by an AND-operator. Interpreting these terms as vague, other connecting operators may be used, like ACCUMulate or OR.

## 5. EVALUATION OF THE NEW IRSTREAM ENGINE AT INEX 2003

With the runs submitted to INEX 2003, two things were to be looked at: First, we wanted to see, whether our interpretation of CAS-topics and thus the differentiation between content and structure constraints would lead to good results compared to those of the other participating retrieval systems. Second, we wanted to get an estimation of how applicable our approach for processing CO-topics is.

Figures 4 and 5 show the recall/precision graphs for IRStream's CAS-run — with strict and generalized quantization — in comparison to all officially submitted retrieval runs. Rank 12 of 38 for strict quantization and rank 10 of 38 for generalized quantization seem promising that the chosen query architecture forms a solid basis for further efforts.

### INEX 2003: second\_scas

quantization: strict; topics: SCAS  
average precision: 0.2277  
rank: 12 (38 official submissions)

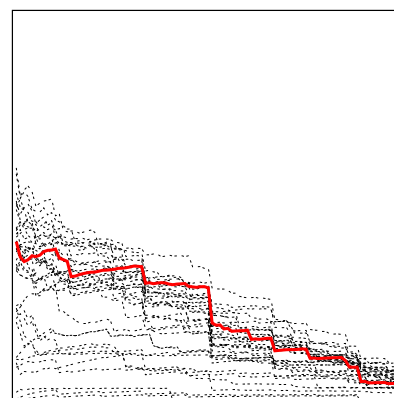


Figure 4: summary CAS strict

The recall/precision graphs for IRStream's CO-run are shown in figures 6 and 7. Rank 10 of 56 for strict and rank 7 of 56 submissions for generalized quantization indicate that further efforts to optimize our approach seem to be worthwhile.

In order to compare the results of IRStream02 and IRStream03 — and thus to evaluate the effect of the system changes — we used the new system to create a retrieval run on the topics of INEX 2002. Since the topic syntax for CAS-topics has changed, only those topics were processed in this run which could be converted to the new syntax. Topics without explicitly stating a target element or those with multiple target elements do not conform to INEX 2003 syntax and thus were omitted.

Figures 8 to 11 show that — especially considering the re-

### INEX 2003: second\_scas

quantization: generalized; topics: SCAS  
average precision: 0.1983  
rank: 10 (38 official submissions)

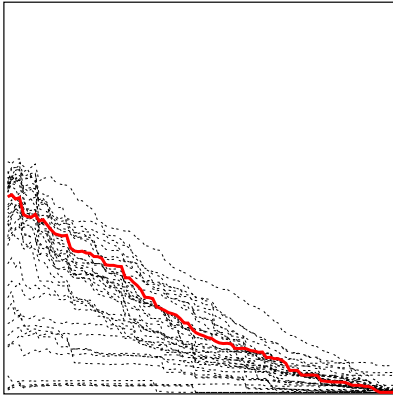


Figure 5: summary CAS generalized

### INEX 2003: \_co\_second

quantization: generalized; topics: CO  
average precision: 0.0717  
rank: 7 (56 official submissions)

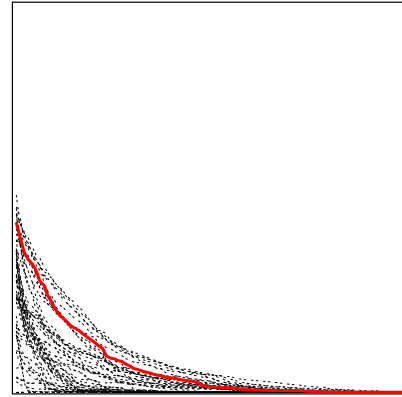


Figure 7: summary CO generalized

### INEX 2003: \_co\_second

quantization: strict; topics: CO  
average precision: 0.0677  
rank: 10 (56 official submissions)

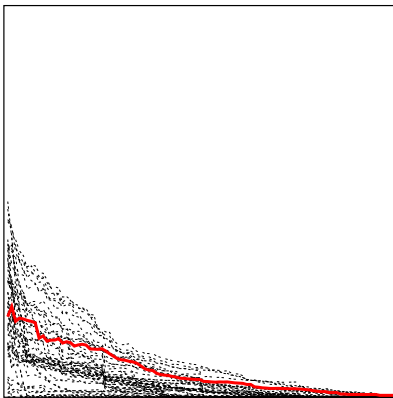


Figure 6: summary CO strict

### IRStream 2002 vs. 2003 quantization: strict; topics: CAS

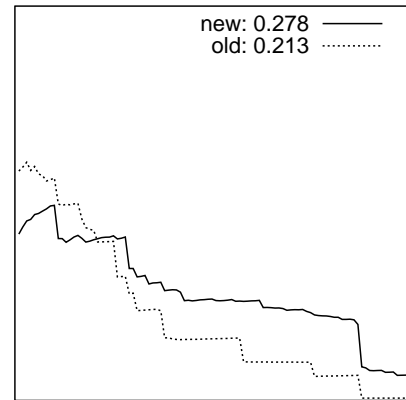


Figure 8: improvement CAS strict

call — the results of IRStream03 are noticeably better than those of IRStream02, which is mainly caused by a more vague interpretation and processing of topic data. Due to the manual optimization of the queries used in IRStream02, its precision at low recall values is slightly better than that of IRStream03, which uses a fully automated query processing.

## 6. CONCLUSION

In this paper, we have presented an improved version of our retrieval system called IRStream, which was successfully used in the context of INEX 2002. The main idea of IRStream is to complement traditional query processing techniques for queries dominated by similarity conditions. The IRStream retrieval engine has been implemented as a prototype in Java on top of an ORDBMS and first experimental results achieved with this prototype are promising.

### IRStream 2002 vs. 2003 quantization: generalized; topics: CAS

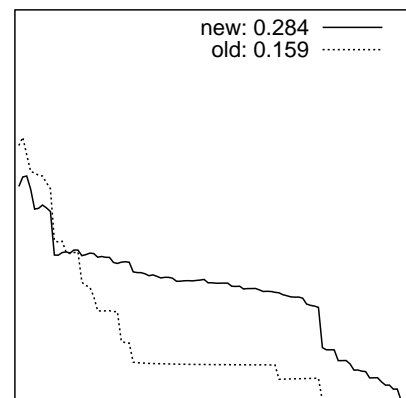


Figure 9: improvement CAS generalized

IRStream 2002 vs. 2003  
quantization: strict; topics: CO

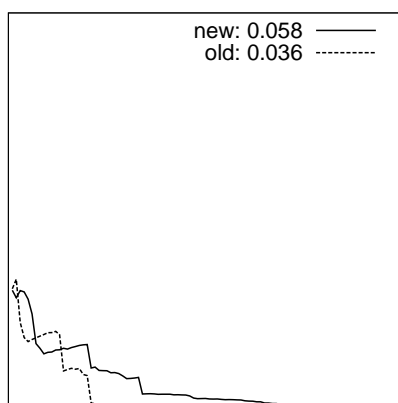


Figure 10: improvement CO strict

IRStream 2002 vs. 2003  
quantization: generalized; topics: CO

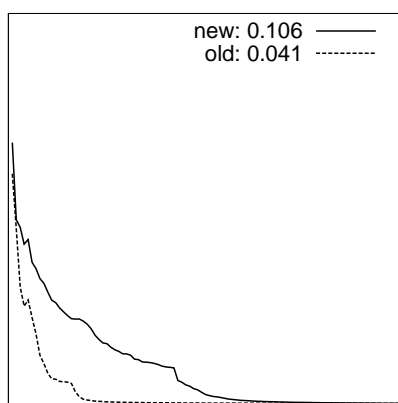


Figure 11: improvement CO generalized

With regard to INEX2003 IRStream was extended and improved in several respects. IRStream now supports automatic query generation as well as the automatic detection of the best fitting result granule for a given query.

In the near future, we will develop a query language for this approach and consider optimization issues regarding the interaction between the underlying ORDBMS and the IRStream system. Last but not least, IRStream should build a good basis for the integration of further query criteria — like context information or domain specific thesauri — into the query execution in order to improve the precision of the system.

## 7. REFERENCES

- [1] R. Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999.
- [2] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. 10th ACM Symposium on Principles of Database Systems: PODS*, pages 102–113, New York, USA, 2001.
- [3] R. Fagin and E. L. Wimmers. A formula for incorporating weights into scoring rules. *Theoretical Computer Science*, 239(2):309–338, 2000.
- [4] N. Fuhr and M. Lalmas. *Initiative for the Evaluation of XML retrieval (INEX)*. Online available: url: <http://inex.is.informatik.uni-duisburg.de:2003>, 2002.
- [5] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *VLDB 2000, Proc. 26th Intl. Conf. on Very Large Data Bases*, pages 419–428, Cairo, Egypt, 2000.
- [6] A. Henrich. Document retrieval facilities for repository-based system development environments. In *Proc. 19th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 101–109, Zürich, Switzerland, 1996.
- [7] A. Henrich and G. Robbert. Combining multimedia retrieval and text retrieval to search structured documents in digital libraries. In *Proc. 1st DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries*, pages 35–40, Zürich, Switzerland, 2000. ERCIM Workshop Proceedings.
- [8] A. Henrich and G. Robbert. POQL<sup>MM</sup>: A query language for structured multimedia documents. In *Proc. 1st Intl. Workshop on Multimedia Data and Document Engineering, MDDE'01*, pages 17–26, Lyon, France, 2001.
- [9] A. Henrich and G. Robbert. A graphical user interface for complex similarity queries on structured multimedia documents. In *Proceedings of the 3rd International Workshop on Multimedia Data and Document Engineering (VLDB Workshop)*, Berlin, Germany, 2003.
- [10] A. Henrich and G. Robbert. RSV-Transfer: An algorithm for similarity queries on structured documents. In *Proceedings of the 9th International Workshop on Multimedia Information Systems (MIS 2003)*, pages 65–74, Ischia, Italy, May 2003.
- [11] G. Kazai, N. Gövert, M. Lalmas, and N. Fuhr. *The INEX evaluation initiative*, pages 279–293. Lecture Notes in Computer Science. Springer, Heidelberg et al., 2003.
- [12] J. H. Lee. Analyses of multiple evidence combination. In *Proc. 20th Annual Intl. ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–276, Philadelphia, PA, USA, 1997.
- [13] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *Proc. 27th Intl. Conf. on Very Large Data Bases*, pages 281–290, Los Altos, USA, 2001.
- [14] U. Pfeifer and S. Pennekamp. Incremental Processing of Vague Queries in Interactive Retrieval Systems. In *Hypertext - Information Retrieval - Multimedia '97: Theorien, Modelle und Implementierungen*, pages 223–235, Dortmund, 1997.