Applying the IRstream Retrieval Engine for Structured Documents to INEX

Andreas Henrich
University of Bayreuth
D-95440 Bayreuth, Germany
andreas.henrich@uni-bayreuth.de

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

ABSTRACT

For a long period of time the research activities in information retrieval have mainly addressed flat text files. Although there have been approaches towards multimedia data and structured data in the past, these topics gain increasing interest today in the context of XML data. To address structured multimedia data, an efficient combination of content-based retrieval for multimedia data, retrieval in meta data and mechanisms which allow to exploit the document structure is needed.

To this end, we propose *IRstream* as a general purpose retrieval service for structured multimedia documents. IRStream is intended as a powerful framework to search for components of arbitrary granularity – ranging from single media objects to complete documents. At this, 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. We describe the significant features of this approach and point out the system architecture. Furthermore, we present the application of IRstream as a retrieval system for XML documents in the context of INEX.

1. MOTIVATION

Today, electronic documents are more than flat text, rather they form a complex structure of different parts. Besides text data, we can find other media types like audio, image, and video. Furthermore, documents can contain meta data concerning the contained media objects, the internal document structure, and the document itself.

To deal with such documents, we need an efficient combination of (1) content based retrieval techniques for text and multimedia data, (2) search mechanisms which can address and exploit the structure of the documents, (3) retrieval in meta data, and (4) traditional retrieval facilities such as fact retrieval or pattern matching. Finally – according to the experiences in the information retrieval community – the retrieval system should yield a ranking based on some type of similarity conditions. In the context of structured multimedia data, the system has to allow for a flexible and precise definition of these similarity conditions.

In the present paper, we propose a stream-oriented approach to process such complex similarity-based queries. The basic idea is to deploy access structures efficiently supporting similarity queries wherever possible. These access structures produce initial streams which can be combined and transferred afterwards. To this end, we use components which combine multiple rankings (usually derived for different ranking criteria) and transfer rankings derived for objects of a certain type to objects of a related type. An important feature of the approach is that it is pull-based, i.e. each stream extracts elements from its input streams only on demand. This can be seen as a lazy evaluation approach, where each input stream is produced only to the extent needed to produce the desired number of elements in the final output stream presented to the user.

Obviously, this approach is not only applicable with structured multimedia documents, but also in the area of structured text documents. Especially the increasing use of XML in digital libraries, product catalogues, scientific data repositories and across the Web encouraged the development of appropriate searching and browsing methods. For this reason, the Initiative for the Evaluation of XML retrieval (INEX) [5] initiated an international, coordinated effort to promote evaluation procedures for content-based XML retrieval. INEX provides an opportunity for participants to evaluate their retrieval methods using uniform scoring procedures and a forum for participating organizations to compare their results. As a participating organization, we applied IRstream to the collection of XML documents provided by INEX. Hereby, we investigated the usability of IRstream for structured text documents.

The rest of the paper is organized as follows: In section 2 we will give a first rough description of our approach. Thereafter we will go into the details of the main components of IRstream in section 3. The concrete architecture of our IRstream implementation is presented in section 4. Section 5 shows how IRstream can be used as a retrieval engine for XML documents in the context of INEX and presents the experiences gained. Finally, section 6 concludes the paper.

2. A FIRST VIEW

A first impression of our approach can be given best by an example. Such an example for a query dominated by

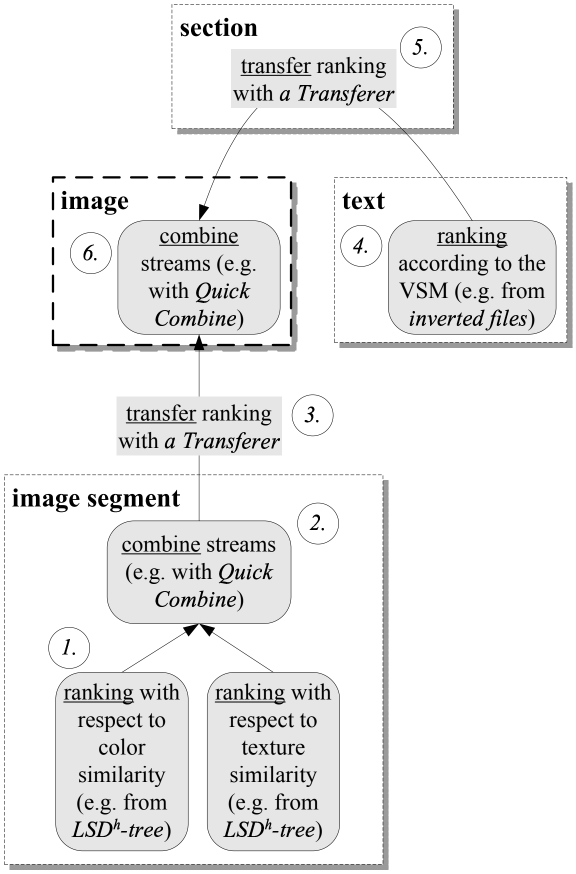

Figure 1: Stream-oriented processing of our example query

ranking conditions might arise when the user is searching for images maintained in structured multimedia documents. Here, the user might be interested in images containing a given logo – i.e. images which contain a segment similar to the given logo – where the text nearby the image is dealing with skiing or winter sports in general. This query contains two ranking conditions: (1) There is a ranking condition for the text in the vicinity of the desired images and (2) a ranking condition for image segments which are required to be similar to a given logo.

Now we assume that the multimedia documents consist of a set of sections. Each section contains images and/or text blocks. Furthermore, each image is associated with several image segments. In this case, our example query searching for images containing a given logo where the text nearby the image is dealing with skiing or winter sports in general can be processed as depicted in figure 1.

First, two different rankings are generated for the image segments delivering these image segments sorted according to their color and texture similarity, respectively, compared to the given logo. To this end, feature vectors representing the color and texture characteristics of each image segment are applied. Comparing these vectors with the given logo, two *retrieval status values* are calculated for each image segment defining the rankings for the color and texture similarity. For the efficient stepwise calculation of these rankings various access structures have been proposed, such as the M-tree, the X-tree or the LSD^h-tree [15, 1, 8]. In figure 1 this part of the query evaluation process is indicated as step 1.

Then the rankings derived for the two criteria have to be combined into a single weighted ranking (step 2). To this end, algorithms such as Fagin's algorithm [2, 3], Nosferatu [14] or Quick-Combine [6] can be deployed.

Now we have derived a combined ranking for the image segments. However, what is needed is a ranking for the images themselves. To derive this ranking, we transfer the ranking for the image segments to the images. To this end, we exploit that each image segment is associated with some type of retrieval status value determining the ranking of the image segments. As a consequence, we can transfer the ranking for the image segments to the images based on these retrieval status values. For example, we can associate the maximum retrieval status value of a related image segment with each image. To implement this transfer of the ranking, we consider the ranking for the image segments one element after another, determine the associated image and calculate the corresponding ranking of the images (step 3). More details of this algorithm will be presented in section 3.3.

Now we have to derive a second ranking for the images with respect to the requirement that the text nearby the image — i.e. in the same section — is dealing with skiing or winter sports in general. To this end, a ranking for the text blocks can, for example, be created via an implementation of the vector space model using inverted files (step 4). Then this ranking has to be transferred from the text blocks to the images in the same section (step 5). Now we have got two rankings for the images: one concerning the "logo criterion" and one concerning the "text in the vicinity criterion". Finally these rankings have to be combined to yield a common ranking for the images (step 6).

3. 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.

3.1 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^h-tree, or by approaches based on inverted files. All these access structures can perform the 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 type of "getNext" method. Hence, the produced streams can be efficiently consumed one element after the other.

3.2 Combiners

Components of this type combine multiple streams providing the same objects ranked with respect to different ranking criteria. Images are an example for 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 over 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 over the retrieval status values in the input streams can be used to derive the overall ranking [4]. Other approaches are based on the ranks of the objects with respect to the single criterion [12, 9]. To calculate such a combined ranking efficient algorithms, such as Fagin's algorithm [2, 3], Nosferatu [14], Quick Combine [6] and J^* [13] can be deployed.

3.3 Transferers

With structured documents, ranking criteria are sometimes not defined for the required objects themselves but for their components or other related objects. An example arises when 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. This transfer of a ranking onto related objects seems to be worth a more in-depth consideration.

Before we can explain the algorithm for the transfer of a ranking, we have to clarify the semantics of this transfer. To this end, we consider a simplified example query where the user is searching for images containing an image segment similar to a given logo. Here the situation is as follows: We have a retrieval status value for the image segments. This value allows to derive a ranking for the image segments. However, we are not interested in a ranking of the image segments but in a ranking of the images. Therefore it is necessary to derive a retrieval status value for each image.

Let $RSV_r(ro)$ be the retrieval status value of object ro (ro for "related object" and RSV_r for the RSV values of "related" objects). In our example ro would be an image segment. Further let $\{ro_{i,1}, ro_{i,2}, \ldots, ro_{i,n_i}\}$ be the set of related objects associated with the "desired object" do_i. In our example this set would contain the image segments associated with the image do_i. Finally let us assume that high RSV values stand for well fitting objects. Then we need a function $\mathcal{F}$ deriving the retrieval status value $RSV_d(do_i)$ from the objects associated with do_i and their RSV values:

$$RSV_d(do_i) \stackrel{\text{def}}{=} \mathcal{F}\left(\begin{array}{l} \langle ro_{i,1}, RSV_r(ro_{i,1})\rangle, \\ \langle ro_{i,2}, RSV_r(ro_{i,2})\rangle, \\ \quad\vdots \\ \langle ro_{i,n_i}, RSV_r(ro_{i,n_i})\rangle \end{array}\right)$$

Examples for meaningful choices for $\mathcal{F}$ are the maximum RSV_r value, the average RSV_r value, a weighted average RSV_r value, or even the minimum RSV_r value.

Now the problem which has to be solved by a transferer can be described as follows: We are concerned with a query which requires a ranking for 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 (*image segments* for example).

We assume that the relationship between these objects is well-defined and can be traversed in both directions. For our example, this means that we can determine the concerned image for an image segment and that we can determine the related image segments for an image. In this situation there will be only one concerned image for each image segment but situations are conceivable where a related object is shared by multiple desired objects. In this case, we get multiple objects of type ot_d.

In addition, we assume there is an input stream yielding a ranking for the objects of type ot_r.

Based on these assumptions, the "transfer algorithm" can proceed as follows. It uses the stream with the ranked objects of type ot_r as input. For the elements from this stream, the concerned object – or objects – of type ot_d are computed traversing the respective relationships. Then the RSV_d values are calculated for these objects of type ot_d according to the desired semantics and the object of type ot_d under consideration is inserted into an auxiliary list maintaining the objects considered so far. In this list, each object is annotated with its RSV_d value. Now the next object of type ot_r from the input stream is considered. If the RSV_r value of this object is smaller than the RSV_d value of the first element in the auxiliary list which has not yet been delivered in the output stream, this first element in the auxiliary list can be delivered in the output stream of the transfer component.

For a more detailed consideration, we have to define the characteristics of the auxiliary list AL. AL maintains pairs $\langle do_i; RSV_d(do_i)\rangle$ with type$(do_i) = ot_d$. These pairs are sorted in descending order with respect to their RSV_d values. For AL the following operations are needed: createAL() creates an empty auxiliary list. getObj(AL, i) yields the object with the i^{th} highest RSV_d value stored in AL. getRSV(AL, i) returns the RSV_d value for the object with the i^{th} highest RSV_d value stored in AL. contains(AL, do_j) checks whether there is an entry for object do_j in AL. insert$(AL, \langle do_l; RSV_d(do_l)\rangle)$ inserts the entry for do_l into AL preserving the sorting with respect to RSV_d – moreover, if other objects with the same RSV_d value are already present in AL, the new object is placed behind these objects in AL. size(AL) returns the number of entries in AL.

Based on these definitions, we can state a class *Transferer* which provides a constructor and a getNext method. This class is given in pseudo-code in figure 2. The attributes which have to be maintained for a transferer comprise the input stream, a definition of the desired relationship between the objects of type ot_r and ot_d, the auxiliary list, a variable o_r which stores the next object of the input stream, and the number of delivered objects.

It has to be mentioned that the *maximum* semantics allows for some simplifications of the presented algorithm. With this semantics, there is no need to calculate RSV_d values in the **foreach** loop, because if there is no entry for o_d in AL, o_r is surly the related object with the highest RSV_r value for o_d. Consequently, $RSV_d(o_d) = RSV_r(o_r)$ holds, and the operation insert $(AL, \langle o_d; RSV_d(o_d)\rangle)$ in the getNext method can be replaced by the more efficient operation insert $(AL, \langle o_d; RSV_r(o_r)\rangle)$.

3.4 Filters

Of course, it must be possible to define filter conditions for all types of objects. With our stream-oriented approach

```
Class Transferer {
  Stream : inputStream;
  RelationshipDef : rel_d; /* desired relationship */
  AuxiliaryList : AL;
  InputObject : o_r; /* next object to be considered */
  Integer : n; /* no. of next object to be delivered */

  constructor(Stream : input, RelationshipDef : rel) {
    inputStream := input;
    rel_d := rel;
    AL := createAL();
    o_r := streamGetNext(inputStream);
    if o_r = ⊥ then exception("empty input stream");
    n := 1;
  }

  getNext() : OutputObject {
    while o_r ≠ ⊥ ∧ (size(AL) < n
                   ∨ RSV_r(o_r) ≥ getRSV(AL, n)) do
      /* consider the next input object o_r */
      SDO := {o_d | ∃rel_d(o_r → o_d)};
        /* all objects which can be reached via
           the desired relationship */
      foreach o_d ∈ SDO do
        if ¬contains(AL, o_d) then
          insert (AL, ⟨o_d; RSV_d(o_d)⟩);
      end /* foreach */;
      o_r := streamGetNext(inputStream);
    end /* while */;
    if o_r = ⊥ ∧ size(AL) < n then
      return ⊥; /* stream exhausted */
    else
      n++;
      return getObj(AL, n − 1);
    end /* if */;
  }
}
```

Figure 2: Class *Transferer* in pseudo code

this means that filter components are needed. 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.

4. 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 3 depicts this architecture.

The core IRstream system — shaded grey in figure 3 — 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.

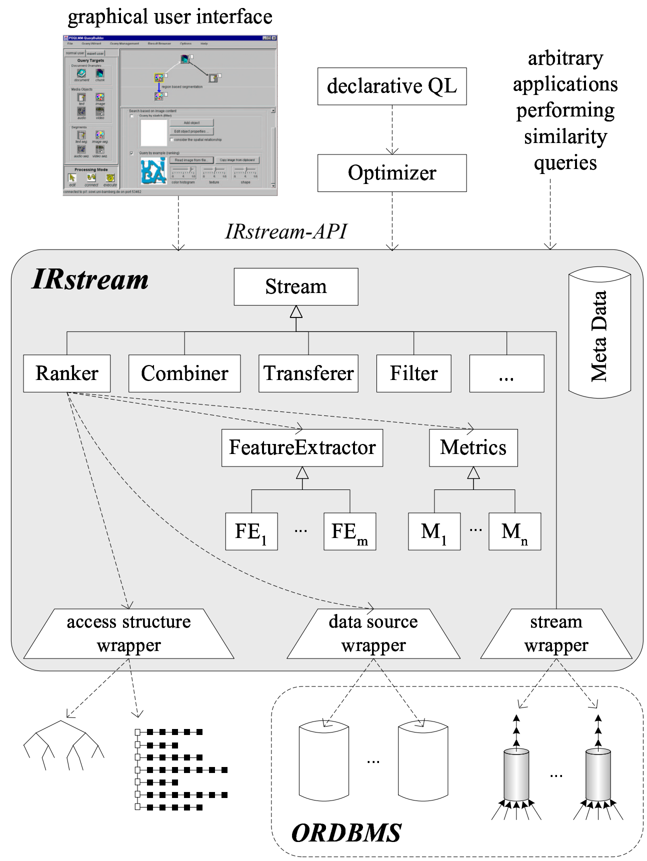

Figure 3: Architecture of the IRstream system

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 of our system maintains meta 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. Here the meta data can e.g. be used to control the query construction in a graphical user interface.

Wrapper

Data source wrappers are needed to attach systems maintaining the objects themselves to our retrieval system. At present, ORDBMSs can be attached via JDBC.

Access structure wrappers can be used to deploy access structures originally not written for our system. For example, we incorporated an LSD^h-tree implementation written in C++ via a corresponding wrapper. In general, this interface should be used to attach access structures which can maintain collections of feature values and perform similarity queries on these values.

Finally, *stream wrappers* can be used to incorporate external stream producers. At present, the text module of the underlying ORDBMS is integrated via a stream wrapper. In contrast to an access structure, such an external stream producer provides not only a ranking but also access to the maintained objects themselves. This means that an external stream producer is aware of the objects themselves, whereas an external access structure does only maintain feature values and associated object references.

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 [10]. 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 $\mathrm{POQL}^{\mathrm{MM}}$ query language [7, 11].

5. IRSTREAM IN THE CONTEXT OF INEX

To assess the applicability of our IRstream approach as a retrieval engine for XML documents, we performed up to sixty retrieval runs on the INEX test collection containing more than ten thousand documents. All these documents where inserted into the ORDBMS underlying our system.

document part	cardinality
journal	124
article	11,993
author	21,902
frontmatter	11,993
body	11,993
backmatter	9,954
section/subsection/...	140,417
paragraph	1,398,494

Table 1: Addressable document parts and their cardinality

To this end, we parsed all documents and decomposed them hierarchically into several parts. Table 1 depicts all document parts and their cardinality. By these means, we can address different granules of the documents in order to support a search concerning the document structure.

Furthermore we implemented a specialized ranker for XML data which internally uses the text retrieval functionality provided by the underlying ORDBMS, and incorporated this ranker into our IRstream retrieval engine. Using this approach, we were able to deal with all sixty topics.

In the following, we point out how the query processing in IRstream is done by means of a typical example topic. To this end, we consider topic 3, which is a so-called *content and structure topic* (CAS):

Title:	information data visualization
Context:	*Keyword*: information data visualization *Document*: large information hierarchies spaces multidimensional data databases
Description:	I am looking for techniques for visualizing large information hierarchies or information spaces.
Narrative:	For a document or document element to be considered relevant, the document (element) has to deal with visualization techniques for data mining or visualization techniques for large textual information spaces or hierarchies. Document/document components describing visualization of any multidimensional data (be it hierarchical or otherwise) are relevant. Documents describing rendering techniques and algorithms are not relevant.

To process topic 3 we used three rankers, three transfers and one combiner. Figure 4 shows the involved components and their interaction for the stream-oriented processing of topic 3 with IRstream.

First we used one ranker to determine a ranking for the document parts of type *frontmatter*, where the attribute *keyword* (tag `<keywd>`) contains terms like *"information data visualization"*. In parallel, we employed two rankers to acquire a ranking for the document parts of type body (tag `<bdy>`) concerning the terms *"information hierarchies"* and *"information techniques"*. The original query text and the addressed document granule are depicted in the boxes of figure 4 named *XML ranker*.

In order to get whole articles as result elements, we used three transferers applying the maximum semantics to map the results of the different streams onto the document type

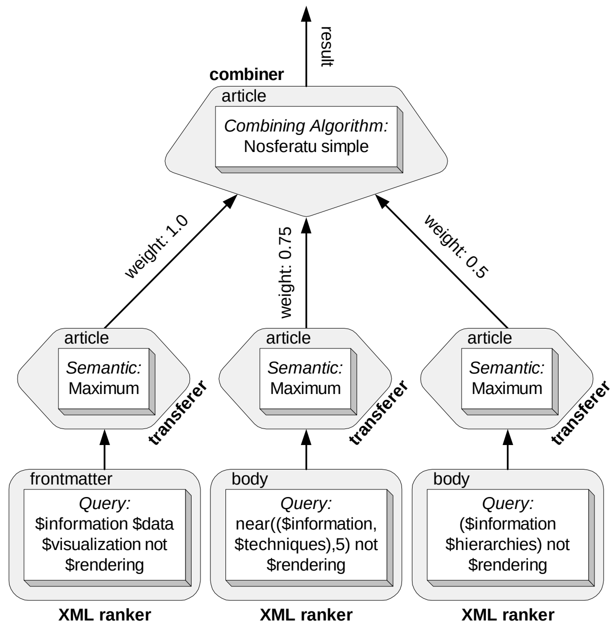

Figure 4: Stream-oriented processing of topic 3

article.

Last but not least, to achieve the final result we used a combiner to merge the ranking of the three incoming streams using the algorithm Nosferatu simple [14]. For the merging of the different input streams, a weight was assigned to each stream in order to control the influence of the different document parts. The weights are noted at the arrows leading from the tranferers to the combiner in figure 4.

For all topics the average response time of the IRstream retrieval engine was about one second. It has to be noted that all query processing has been performed with a first IRstream prototype. This prototype implemented in Java is by no means optimized.

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented an approach for the stream-oriented processing of complex similarity queries. The approach is intended to complement traditional query processing techniques for queries dominated by similarity conditions. The approach has been implemented as a prototype in Java on top of an ORDBMS and first experimental results achieved with this prototype are promising.

In the near future, we will address the optimization of the prototype implementation and perform experiments with larger test collections. Furthermore, 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 — into the query execution in order to improve the precision of the system.

7. REFERENCES

[1] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An index structure for high-dimensional data. In *VLDB'96, Proc. 22th Intl. Conf. on Very Large Data Bases*, pages 28–39, Mumbai, India, 1996.

[2] R. Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999.

[3] 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.

[4] R. Fagin and E. L. Wimmers. A formula for incorporating weights into scoring rules. *Theoretical Computer Science*, 239(2):309–338, 2000.

[5] N. Fuhr, M. Lalmas, G. Kazai, and N. Gvert. *Initiative for the Evaluation of XML retrieval (INEX)*. Online available: url: http://qmir.dcs.qmul.ac.uk/inex/, 2002.

[6] 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.

[7] 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.

[8] A. Henrich. The LSD^h-tree: An access structure for feature vectors. In *Proc. 14th Intl. Conf. on Data Engineering, Orlando, USA*, pages 362–369, 1998.

[9] 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.

[10] A. Henrich and G. Robbert. An end user retrieval interface for structured multimedia documents. In *Proc. 7th Workshop on Multimedia Information Systems, MIS'01*, pages 71–80, Capri, Italy, 2001.

[11] A. Henrich and G. Robbert. POQL^{MM}: 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.

[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.

[15] P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with M-trees. *VLDB Journal*, 7(4):275–293, 1998.