Towards a Scalable Networked Retrieval System for Searching Multimedia Databases

Christoph Baumgarten, Klaus Meyer-Wegener *
Dresden University of Technology, Germany

Abstract

In this paper the architecture of a *d*istributed and *s*calable *m*ultimedia *i*nformation retrieva*l* s*y*stem (D*smily*) is described. The system consists of hierarchically organized networked nodes and is designed to integrate existing dynamic multimedia databases. The document ranking process as well as the preselection of databases to be searched, both tasks are based on a probabilistic model for distributed retrieval.

1 Introduction

The Internet provides access to a large amount of data which is growing from day to day. Most of the data are simple text documents, but the fraction of multimedia data like image, audio and video documents is increasing rapidly. Hence, for users, it is getting more and more difficult to find documents containing relevant information. This is true especially for multimedia documents, because often one cannot clearly decide whether a document is relevant or non-relevant. That means, traditional boolean search techniques are not sufficient in this case and therefore have to be replaced by search techniques which enable the handling of 'vague' queries. Such techiques have been developed within the research area of Information Retrieval (IR) during the last decades. However, most of them are restricted to non-distributed text databases. Extending them to be applicable to distributed multimedia databases is still a challenge.

Our aim is therefore the design of D*smily*, a networked IR system, which provides access to an arbitrary number of widely distributed multimedia databases. These databases contain documents (or in other words, retrievable items) of arbitrary content, format and size, they are autonomously managed and may behave highly dynamic. By using the notion 'multimedia database' we do not only refer to modern multimedia database systems but also to traditional database systems or even simple file systems. An example for a modern multimedia database system is the media object storage server KANGAROO that is under development at Dresden University of Technology [KMM94].

Databases might offer their own retrieval functions. Relational databases for example usually provide SQL, simple file systems have some sort of `grep` command. Such functions are not used by D*smily*. Instead, each database is supplied with a retrieval functionality satisfying certain global requirements.

D*smily* is hierarchically build, communication between the D*smily* nodes is planned to be realized using the CORBA middleware [Obj91]. It is intended to adopt concepts and modules from the SPIDER information retrieval system [Sch93] [KS96], which is currently under development at the Swiss Federal Institute of Technology (ETH) in Zurich.

Architectures that show some similarity to the one presented here, include the MEDOC approach [BDG96], Harvest [Bow94], gGlOSS [GGM95] as well as the CAFE architecture [CN96].

*E-mail: {baumgart, kmw}@is2201.inf.tu-dresden.de

2 Architectural concepts

The architecture of D*smily* is based on a probabilistic model for distributed IR, which is discussed in detail in [Bau97a]. The model assumes the underlying document collection to be partitioned into disjunctive *subcollections*, which again may be combined to superior subcollections. This leads to a hierarchy of subcollections. No superior unit influences the assignment of a document (or subcollection) to a subcollection.

IR models usually specify methods for ranking documents with respect to a given query. The higher a document is ranked, the more relevant the document is considered to be. These ranking algorithms rely on statistical information gathered from the underlying document collection. While in a non-distributed setting, the entire collection is considered, the model employed here manages with subcollection-wide statistical information. Hence, in order to allow the collection of this information, not only queries and documents, but also subcollections have to be indexed by a set of *indexing features* (also called a *description*) taken from a certain *indexing vocabulary*. Corresponding indexing methods can be chosen subcollection-specific, i.e. the elements of a subcollection may be indexed using a vocabulary different from the one employed for indexing the elements of another subcollection. This way, the integration of autonomously managed media document collections can be supported. [Bau97b] presents an extension of the model, which enables the appropriate handling of *multi*media documents, i.e. documents, which compose of passages that have to be indexed using different methods mapping into different vocabularies.

This desirable property of the model is completed by the possibility to arbitrarily chose IR methods from a set of possible probabilistic methods originally designed for non-distributed IR and to assign them to the different subcollections. Having individually ranked the elements of the subcollections using the assigned non-distributed IR method, the resulting local ranking lists are combined and merged into a final ranking list such that the result is optimum with respect to the *Probability Ranking Principle* [Rob77]. A ranking list contains the documents together with their scores or retrieval status values (RSVs), respectively.

Further properties of the model are:

- The retrieval process can be automatically limited to parts of the distributed document collections while minimizing the loss of relevant documents.
- Relevance feedback (given either on documents or subcollections) is utilized — an aspect, which is not further considered in this paper.
- An unlimited growth of the data volume is supported by the model.

But how should a distributed IR system look like, which exploits the full potential of the probabilistic model? In particular, how can providers of multimedia document collections be given the opportunity to make their data searchable while keeping as much autonomy as possible (for example using own indexing and ranking methods)?

Our proposed architecture composes of so called D*smily nodes* (Figure 1). Each node provides a set of external interfaces to certain objects, which are under the administration of a single provider. D*smily* nodes are hierarchically organized: While nodes on layer 1 of the hierarchy manage the inclusion of existing databases, nodes on higher layers give access to subordinated nodes.

An *external interface* of a node is given in terms of CORBA's IDL (interface definition language) and provides the functionality that will be provided by an object of the node to other nodes. It defines the object's attributes as well as operations (operations are implemented by methods). A corresponding *internal interface* offer additional functionality, which manage the communication between the objects *within* the node and are assumed to be not visible to the objects of other nodes. This internal interface can be realized either through additional methods in the class implementing the

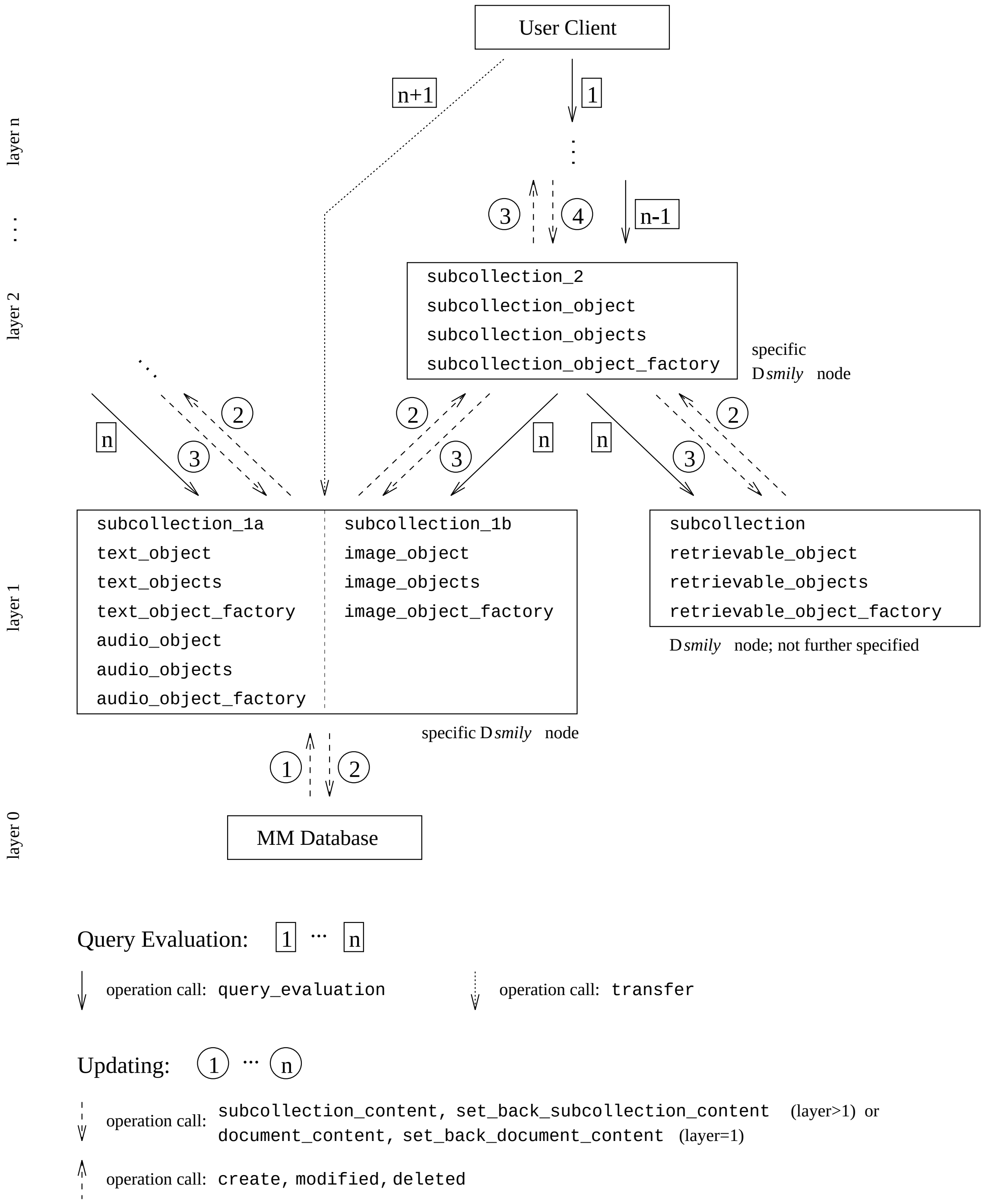

Figure 1: The D*smily* architecture.

external interface or through an interface that the external interface is derived from. The advantage of the first variant in comparison to the second is that the internal interface is really hidden from other nodes. The second variant on the other hand simplifies the distribution of a node's components to different processing sites — a useful feature, since a node can become rather extensive.

In the following, the set of external interfaces of an arbitrary node on layer i is given. The external interfaces of a *specific* node are to be derived from this set by the node's provider. At this, it is possible to derive more than one interface from one of the listed *general external interfaces*. Examples will be given in section 3.

```
typedef any client_id_type;

interface retrievable_object{
// Interface defining a media or subcollection object.

   void transfer(in client_id_type client_id);
   // Operation for transferring the content of an object to the user client with the identifier client_id.
   // Further investigation of multimedia aspects required here.  Usually not implemented for subcollection
   // objects.

};

typedef any id_type;

interface retrievable_objects{
// Interface to the set of all retrievable objects.  An interface derived from retrievable_objects has exactly
// one corresponding interface derived from retrievable_object.

   void modified(in id_type id);
   // Marks object with the identifier id as modified.  If the object does not exist, the call is ignored.

   void delete(in id_type id);
   // Deletes object with the identifier id together with the object description.  Updates the content
   // alteration of the object´s subcollection.  If the object does not exist, the call is ignored.
};

interface retrievable_object_factory{
// Factory for creating retrievable objects.  An interface derived from retrievable_object_factory has
// exactly one corresponding interface derived from retrievable_object.

   retrievable_object create(in id_type id);
   // Creates a retrievable object from a given object identifier.  The object is marked as new.  Decides
   // on the subcollection where the object is assigned to.
};

typedef any query_type;
typedef any content_passage_type;

typedef struct content_struct{
           content_passage_type added_content_passage;
           content_passage_type removed_content_passage;
         } content_type;

typedef struct ranking_entry_struct{
           retrievable_object ranked_object;
           double rsv;
         } ranking_entry_type;

typedef sequence<ranking_entry_type> ranking_type;

interface subcollection{
// Interface to a set of retrievable objects, which form a subcollection.  An interface derived from
// subcollection may correspond to more than one interface derived from retrievable_object.
```

```
    readonly attribute content_type subcollection_content;
    // Yields the subcollection´s content alteration.  The content alteration is derived from the summary
    // alterations of the retrievable objects in the subcollection.

    void set_back_subcollection_content();
    // The subcollection´s content alteration is set to zero.

    ranking_type query_evaluation(in query_type query,
                                  in double subcollection_rsv,
                                  inout long nof_desired_documents);
    // Produces a ranking list with respect to the query.  nof_desired_documents gives the desired number of
    // documents contained by the ranking list.  The subcollection´s retrieval status value (rsv) biases the
    // locally computed RSV´s.
};
```

Retrievable objects are defined by the `retrievable_object` interface. A retrievable object *corresponds* to a *retrievable item*, i.e. a document or a subcollection that is provided by a subordinated entity. There is *no equivalence* between a retrievable object and the corresponding retrievable item. Subordinated entities are either databases or nodes from layer $i-1$.

Having added a retrievable item to a subordinated entity, the creation of a new retrievable object is triggered by the subordinated entity calling the `create` operation offered by the `retrievable_object_factory`. This operation is also responsible for assigning the created object to a subcollection. As mentioned before, a subcollection contains retrievable objects with descriptions taken from the same indexing vocabulary. It can be accessed through the `subcollection` interface. Having modified a retrievable item, the subordinated entity uses the `modified` operation from the `retrievable_objects` interface in order to appropriately mark the corresponding retrievable object. Both operations, `create` as well as `modified`, require the retrievable item's identifier as input parameter `id`; if the retrievable item is a document, `id` will most likely be a database-specific identifier, if the retrievable item is a subcollection, the identifier of the object representing the subcollection may be used.

The `retrievable_objects` interface is assigned a *synchronizer*[1] that frequently checks the retrievable objects whether they have been marked as modified or as just created. For each marked object, the corresponding *alteration* to the *object description* and *object summary* is computed applying the *indexing method* and the *summary method* attached to the object. The determined description alteration implies the modification (or generation) of the object's description. This description is stored with the subcollection that contains the marked object: the `subcollection` interface is assigned an IR server, that maintains object descriptions as well as the statistical information gathered from these descriptions. Here, the use of the SPIDER IR server is intended. The alteration to the summary of the marked object is also stored with the subcollection. Since the state of the subcollection itself has changed, the synchronizer marks the retrievable object of the superior node that represents the subcollection using the `modified` operation. In a final step, the marked retrievable object's are unmarked by the synchronizer.

Combining the summary alterations of the objects collected in a subcollection yields the alteration to the *content representation* of the subcollection. The passages that have been added to as well as the passages that have been removed from the content representation, both are provided by the `subcollection_content` attribute of the `subcollection` interface. The indexing and summary methods of the superior node, where the subcollection is represented as a retrievable object, depend on the alteration to the object's content and hence, have to read `subcollection_content`. The type

[1] The synchronizer concept has been adopted from the SPIDER IR system. It has been develpoed in order to support the retrieval in a highly dynamic database. The transaction model underlying the synchronizer relaxes the isolation (serializability) requirement of the ACID principle. This is possible because casual 'dirty reads' do not seriously affect the retrieval effectiveness [KS96].

content_type of this attribute therefore has to be adjusted to the type requirements of the depending methods. By maintaining the content alteration instead of the entire content representation of the subcollection, no extensive storage capacities are required. As a consequence, however, the mentioned indexing and summary methods have to work incrementally. The content alteration (and thus all the summary alterations stored with the subcollection) can be removed after having been read using set_back_subcollection_content.

Of course, databases have to provide a document_content attribute and a set_back_document_content operation similar to subcollection_content and set_back_subcollection_content, such that the indexing and summary methods of the layer-1-nodes are able to access the content alterations of the databases' documents.

What has not been considered so far is the deletion of retrievable items. A subordinated entity that intends to delete one of its retrievable items first empties the item. This implies an alteration to the item's content representation. In a next step, the delete operation from retrievable_objects is applied. As with modified, the retrievable item's identifier is used as the input parameter id. The method implementing the operation can be considered as a simplification of the algorithm frequently performed by the synchronizer: The description of the retrievable object determined by id is removed from the subcollection that contains the object; the object's summary alteration is stored with the subcollection. (Note that the computation of the summary alteration makes use of the alteration to the retrievable item's content representation provided either by subcollection_content or document_content. Since this attribute is attached to the retrievable item, the item may not be removed before calling delete.) Finally, the object itself is deleted. As a consequence, the object representing the subcollection is marked as modified. The call of delete smooth the way for removing the empty retrievable item together with the alteration to the item's content representation.

One could think of implementing the summary method through the indexing method. A subcollection's content is then represented by the set of those indexing features that have been used in order to index the objects within the subcollection. This content representation can be even directly used as the subcollection's description (compare e.g. [CZC95]). Unfortunately, the description now depends on the subcollection-specific indexing vocabulary. To give an example, let us consider two subcollections containing text objects. While the objects of the first subcollection have been indexed using a word stemming algorithm, the description of the objects of the second algorithm result from an n-gram indexing algorithm (without prior stemming). Taking the concatenations of the text object descriptions as the descriptions of the subcollections, both subcollections are not comparable any longer, since their descriptions are taken from different indexing vocabularies. Hence, the subcollections may not be element of the same superior subcollection. This problem does not occur with our approach: the subcollections' content representations are given as concatenations of text summaries, where the words are not stemmed. The subcollections are therefore comparable and can be indexed selecting e.g. a stemming algorithm possibly different from the one employed to index the objects of the first subcollection.

The operation query_evaluation offered by the subcollection interface ranks the documents that are (directly or indirectly) included by the subcollection with respect to a query. A subcollection-specific ranking algorithm is used in accordance with the underlying probabilistic model. If the node is the hierarchy's root node, query_evaluation is called by a user client, if the node is located on a deeper layer, the operation is triggered by a superior query_evaluation operation. In general, the query may be of arbitrary type. However, query_evaluation checks in a first step, whether a query object of type retrievable_object can be generated from the query. If this is possible, the indexing operation correlated to retrievable_object can be applied and yields a query description comparable to the descriptions of the objects in the subcollection.

In the following, the method implementing the query_evaluation operation is shown in pseudo code:

```
ranking_type query_evaluation(in query_type query,
                              in double subcollection_rsv,
                              inout long nof_desired_documents){
  [find a factory operation that is able to create query_object of type retrievable_object from query]
  if [creation of query_object is not possible]{
    nof_desired_documents = 0
    return ∅
  }else{
    [Create query_object]
    query_description = [index query_object]
    local_ranking = [rank retrievable objects of the subcollection w.r.t. query_description]
    [shorten local_ranking w.r.t. nof_desired_documents according to the probabilistic model]
    [change nof_desired_documents if necessary]
    if [layer 1 is current layer]
      return local_ranking
    else{
      forall [entry_j ∈ local_ranking] do_in_parallel{
        [determine the subcollection_j interface correlated to entry_j.ranked_object]
        [let subcollection_rsv bias entry_j.rsv according to the probabilistic model]
        [determine nof_desired_documents_j according to the probabilistic model]
        ranking_j = subcollection_j → query_evaluation(query,
                                                       entry_j.rsv,
                                                       nof_desired_objects_j)
      }
      return [merge all ranking_j according to the computed RSVs]
    }
  }
}
```

3 Some node examples

Considering a node that includes a database containing text, image and audio documents, one could think of the following derivations of the general external interfaces presented in section 2:

```
interface text_object:retrievable_object{...};
interface text_objects:retrievable_objects{...};
interface text_object_factory:retrievable_object_factory{...};

interface image_object:retrievable_object{...};
interface image_objects:retrievable_objects{...};
interface image_object_factory:retrievable_object_factory{...};

interface audio_object:retrievable_object{...};
interface audio_objects:retrievable_objects{...};
interface audio_object_factory:retrievable_object_factory{...};

interface subcollection_1a:subcollection{...};
interface subcollection_1b:subcollection{...};
```

While subcollection_1a includes all retrievable objects, which are indexed using some sort of text features taken from the same indexing vocabulary (here all objects of type text_object and audio_object), subcollection_1b corresponds with image_object providing objects that are indexed by features taken from a different vocabulary. The employed indexing method could for example be similar to the one described in [Lor96], which yields as a description a set of numbers extracted from polygons identified in the image.

Note that the decision on assigning a certain retrievable object to a subcollection depends on the implementation of the `create` operation and thus, on the provider of the node. It is of course also possible to organize objects with descriptions from the same indexing vocabulary in different subcollections (e.g. for content reasons).

Our next example is a node on layer $i > 1$. Here, we only have one subcollection (namely `subcollection_2`) collecting subordinated subcollections, where the corresponding `subcollection_content` operations all map into the same `content_type` (the input type of the indexing and summary method, respectively, correlated to `subcollection_object`):

```
interface subcollection_object:retrievable_object{...};
interface subcollection_object_factory:retrievable_object_factory{...};
interface subcollection_2:subcollection{...};
```

4 Outlook

We have outlined the architectural concepts of D*smily* , a distributed IR system, that is able to provide access to an arbitrary number of dynamic multimedia databases. The system consists of autonomously managed hierarchically organized nodes, which communicate via CORBA. The general design of the nodes' (external) interfaces has been given in terms of the CORBA interface description language.

The functioning of the components of a node has been roughly explained; here, further refinement is required. We will define the components' interfaces (internal interfaces) during a next step that exactly specify the inner-node-communication.

Acknowledgement: This work was done at the graduate college "Tools for the Effective Use of Parallel and Distributed Computing Systems" at Dresden University of Technology funded by the German Research Council (DFG) as well as the Freistaat Sachsen. The authors would like to thank Ulrich Marder, Günter Robbert, Wolfgang Schulze and Henrike Süß for the constructive discussions on the D*smily* architecture.

References

[Bau97a] C. Baumgarten. A Probabilistic Model for Distributed Information Retrieval. In *Proceedings of the 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval (Philadelphia)*, 1997.

[Bau97b] C. Baumgarten. Probabilistische Modellierung der effizienten Informationssuche in veteilten multimedialen Dokumentbeständen durch Einschränkung des Suchraums. In *To be presented at the Hypertext – Information Retrieval – Multimedia Conf. (HIM '97, Dortmund, Germany)*, 1997.

[BDG96] D. Boles, M. Dreger, and K. Großjohann. MeDoc Information Broker. Harnessing the Information in Literature and Full Text Databases. In *Proceedings of the SIGIR'96 Workshop Networked Information Retrieval (Zurich, Switzerland)*, 1996.

[Bow94] C. M. Bowman et al. Customizable Discovery Harvest: A Scalable and Access System. Technical Report CU-CS-732-94, University of Colorado, 1994.

[CN96] G. Crowder and C. Nicholas. Resource Selection in CAFE: an Architecture for Networked Information Retrieval. In *Proceedings of the SIGIR'96 Workshop Networked Information Retrieval (Zurich, Switzerland)*, 1996.

[CZC95] J. P. Callan, L. Zhihong, and W. B. Croft. Searching Distributed Collections With Inference Networks. In *Proceedings of the 18th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 1995.

[GGM95] L. Gravano and H. Garcia-Molina. Generalizing GlOSS to Vector-Space Databases and Broker Hierarchies. In *Proceedings of the 21th VLDB Conf.*, 1995.

[KMM94] R. Käckenhoff, D. Merten, and K. Meyer-Wegener. MOSS as Multimedia Object Server Extended Summary. In R. Steinmetz, editor, *Multimedia: Advanced Teleservices and High Speed Communication Architectures, Proc. 2nd Int. Workshop - IWACA '94 (Heidelberg, Sept. 26-28, 1994), Lecture Notes in Computer Science vol. 868*, Berlin, 1994. Springer-Verlag.

[KS96] D. Knaus and P. Schäuble. The System Architecture and the Transaction Concept of the SPIDER Information Retrieval System. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 1996.

[Lor96] O. Lorenz. *Automatic Indexing of Line Drawings for Content Based Information Retrieval.* PhD thesis, Swiss Federal Institute of Technology, Zurich, 1996.

[Obj91] Object Management Group. The Common Object Request Broker: Architecture and Specification, December 1991.

[Rob77] S. E. Robertson. The Probability Ranking Principle in IR. *Journal of Documentation*, 33(4), 1977.

[Sch93] P. Schäuble. SPIDER: A Multiuser Information Retrieval System for Semistructured and Dynamic Data. In *Proceedings of the 16th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 1993.