An Object-Oriented Framework for Developing Information Retrieval Applications on Digital Libraries

Joemon M. Jose[1], David G. Hendry[2], and David J. Harper[2]

[1] University of Glasgow, Glasgow G12 8QQ, Scotland
jj@dcs.gla.ac.uk
[2] The Robert Gordon University, Aberdeen, AB25 1HG, Scotland
{dgh,djh}@scms.rgu.ac.uk

Abstract. Design and development of information retrieval (IR) systems is a complex and expensive process. In the process of building an IR system, developers encounter a large design space: on the one hand, they have to choose from a plethora of IR techniques developed due to the significant advances made in theoretical and experimental research in information retrieval; on the other hand, many of these techniques need to be tailored to the particular application in hand. Hence, an IR system implementation tool which allows exploration and composition of different system strategies is very valuable to build effective and efficient IR systems. To build such a tool, we need to develop modular and extensible IR components such that these components can be used in a mix and match fashion.This paper describes an extensible and modular framework which can be used to construct IR applications.

1 Introduction

Rapid advancement of technology is instrumental in producing massive amounts of digital information. Researchers in databases and information retrieval have developed sophisticated techniques to deal with such digital libraries. Information retrieval research has been developed a range of techniques for efficiently and effectively storing and retrieving information items based on their text content [12, 3]. Research in databases developed sophisticated techniques for managing structured documents. However, digital libraries such as world-wide-web, office document collections, multimedia collections etc., contain complex documents. That is, documents that contain structured and unstructured parts and, in addition, may be composed of different media types. Many such libraries may contain heterogeneous documents as well. Retrieval applications from such libraries need a combination of information retrieval and database functionalities.

It has been observed that many powerful IR techniques that have been developed have failed to reach application developers mainly due to the cost involved in the implementation[5]. Significant advances made in theoretical and experimental research in information retrieval have many implications for system design. Different application scenarios put different demands on individual

components and the optimal choice of an implementation depends on many parameters. An IR system development tool should support exploration of different designs and implementation strategies. To build such a tool, one needs to decompose and modularise an IR system functionally and provide a mechanism to use these modules in a mix and match form.

In this paper, we describe an object-oriented framework that have been developed for the development of new and emerging IR applications. This framework is based on the philosophy of the ECLAIR class library [5]. The objective for building such a system is two fold: one to develop an extensible and flexible IR architecture; two, to achieve the integration of database and information retrieval functionalities. In the following section, we shall briefly review the process of information retrieval. Followed to that we shall introduce the framework.

2 Information Retrieval

Information retrieval is concerned with identifying documents in a collection that best match a description of a searcher's information need and are likely to contain the information one is looking for. Central to any effective retrieval system is the identification and representation of document content, the acquisition and representation of the information need, and the specification of a matching function that selects relevant documents based on these representations.

Indexing is the process of deriving relevant features from the document to characterise it and is performed to support the computation of relevance values. In general terms, it is the process of identification and selection of features (or signals) that are considered as the representative characteristics of the document content. In the simplest, and the most common representation, each document is described by a set of keywords. This process involves removing most commonly occurring terms (stop-word removal) and conflation of terms (stemming), generally known as the normalisation process.

The output of the indexing process is a set of indexing features. These indexing features need to be organised on storage devices to enable efficient search and retrieval. Most often, these features are stored in a data structure like an inverted index. A retrieval module makes use of this access structure to compute efficiently the similarity between a document and a query. The result of this operation is a set of documents ranked in decreasing order of their similarity to the query. Most retrieval systems require the representation of queries in the same format in which documents are represented, and hence have to be processed (indexed) in the same way as a document.

A retrieval model specifies the details of the document representation, the query representation, and the matching function. Various retrieval models represent different strategies for making assertions about the relevance of a document to a query. Depending on the operational aspects related to the choice of indexing language and also depending on the time space trade-off a number of implementation strategies can be followed for any of the above retrieval models. Each has different performance characteristics both in terms of retrieval effec-

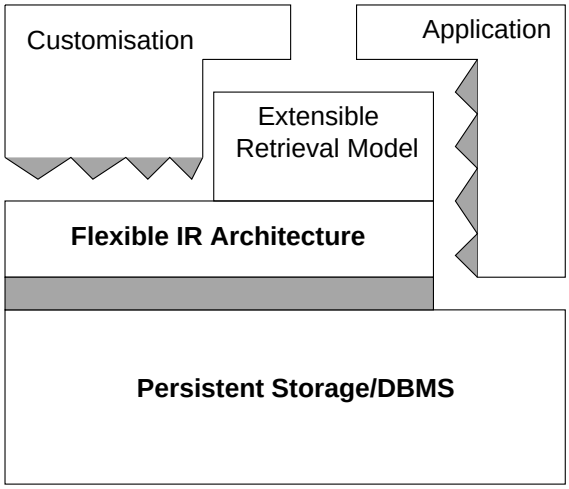

Fig. 1. Implementation architecture of the FLAIR system

tiveness and computational requirements. These left the application developer with a number of choices to make in building an IR system.

In comparison to traditional full text search systems, new digital library applications differs in many respects. One aspect is the structure of the documents that need to be dealt with. Second issue is the possible heterogeneity of underlying documents. Third issue is that for effective retrieval one has to apply different retrieval mechanisms to different components [7, 9]. Moreover, many such applications require functionalities of DBMS as well [5].

We addressed these issues by developing an extensible and modular architecture called FLAIR. In the following, we shall introduce the FLAIR system.

3 FLAIR: Flexible Architecture for Information Retrieval

The basic idea is to develop an extensible retrieval framework so that application developers can build systems easily by using the framework and extending its components. FLAIR provides abstraction and functional modularisation of the information retrieval processes. In addition, FLAIR provides a light weight data model for the representation of the documents. It also integrates the exact and inexact match retrieval in one system.

Figure 1 illustrates the FLAIR implementation architecture. In this scheme, we implement an IR framework on top of a persistent storage. An extensible design is most easily realised using an object-oriented programming language and most OODBMSs support one or more object-oriented programming languages. The underlying OODBMS used in implementing the framework can be used to implement applications and extensions to the framework. By using object-oriented paradigm, the framework can be adapted for applications that use it,

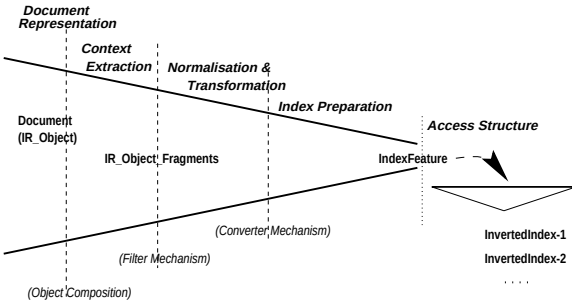

Fig. 2. Conceptual View of an IR System Building Process

and customised as needed. The architecture is flexible in a sense that it can be integrated with any kind of persistent storage. For the current implementation, we used ObjectStore OODBMS and C++ as the programming language. One important feature of FLAIR is the built in extensible retrieval model based on the Dempster-Shafer theory [8]. It allows seamless integration of exact and similarity matching thus providing integration of database and IR functionalities.

FLAIR is designed and implemented using object-oriented methods such as the *scenarios of use* [1] and the *design patterns* techniques [4]. A series of mock scenarios was constructed to identify the abstractions needed. Design patterns have been used to describe the components of the class library and also to capture the static and dynamic structure of solutions.

4 FLAIR Process Model

Building indexes is a main activity in the development of an information retrieval system. In this, documents or their components have to undergo a series of normalisations (like tokenising, stemming, stop-word removal) before being transformed into a set of indexing features. We view this activity as a distillation process applied on a set of documents. That is, in the course of the system building, a document, (and thereby its components), passes through many stages of transformation before being converted into an index structure. This view is depicted in Figure 2.

An IR process starts with the representation of documents. The distillation starts with extraction of the various components of the documents along with their context for indexing, deriving indexing features from these components, normalising and representing these features in appropriate access structures.

The result of this distillation view is that we were able to abstract and modularise the IR system building process. IR design process abstracted into the FLAIR systems is based on this distillation view.

4.1 Abstracting IR Process

We have developed an abstraction for information retrieval process which is shown in Figure 3 . The major processes of IR modelled in the FLAIR system are: modelling of retrievable objects, extraction of components, normalising (or filtering) these components, conversion of these normalised components into indexing features, representation of these features for efficient access, query modelling, query feature extraction, query normalisation, matching (i.e. comparison of query features with document features) and the result generation.

IRObjects represent retrievable documents in the system. IRObjects are subject to a process of indexing (which generates indexing features). Indexing is achieved by a combination of three types of action; Extractor, for extracting components along with their context; Filters, for applying various filtering (normalisation) algorithms like stemming, stop word removal etc.; and Converters for converting a set of normalised features into the form of index features. The class InvertedIndex provides an abstraction for the inverted index access structure in which various kinds of indexing features are organised for efficient access. IRQuery models a searcher's query and is subjected to the same normalisation activity applied to the IRObjects when constructing the indexes for the representation of the documents. The resulting features are stored in instances of a class called QueryRep. A Matcher takes each QueryRep and computes similarity values for each document by comparing them to the access structure (InvertedIndex). The resulting document number and the corresponding similarity values are stored in a class called IRResult. The major activities involved are coordinated in a class called IRModel. IRModel abstracts the particular implementation of a retrieval model, generally known as a retrieval engine.

In new IR applications, it is possible to have multiple index structures (InvertedIndexes) in one retrieval system. This may be the result of applying various retrieval models to various components of the document. Similarly, a query can have multiple components and as a result of these there will be multiple query representatives (QueryReps) and results (IRResults). FLAIR can support these and also provides mechanisms to combine these component results into a final overall result. In the following, we shall introduce the document modelling scheme in FLAIR.

5 Document Modelling

Document abstraction, that is the specification of the retrievable objects and their operations, is needed to accommodate the full range of document types which may be simple textual documents, structured documents, or complex multimedia documents. For this purpose, we introduced a light-weight data model which allows to treat documents and their components uniformly. During the course of building a system, as shown in the distillation view 2, a document or part of it will be transformed into various forms of data until we derive indexing features of interest. A uniform interface for all components of a document will

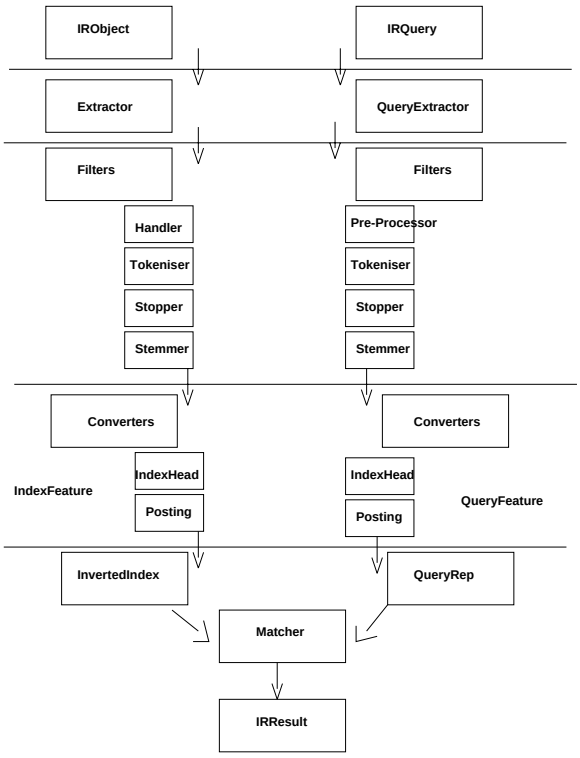

Fig. 3. Main Process in an IR system

enhance the flexibility of the system. In addition, it permits representation of diverse and complex documents.

The model consists of two parts; a hierarchical data typing scheme and a meta-information table. The hierarchical data typing scheme provides a number of ready made data types which are helpful for the content modelling. The meta-information table provides a mechanism to combine these data types to model complex documents.

The data model hierarchy is shown in the Figure 4 based on the *Composite* pattern [4, p. 163]. In this scheme, a document is an aggregation of complex and/or atomic components. This data model composes documents into tree structures of their components. We provide data types to represent atomic and composite components; composite types represent group of atomic ones. The class `DataElement` constitutes the root of the hierarchy. Common functionalities needed for both atomic and complex types are defined in the class DataElement which provides a common interface. Specific implementations of these functionalities are provided in the derived classes. Concrete atomic classes derived from the `DataElement` are `IntegerDE, StringDE, PointDE` and

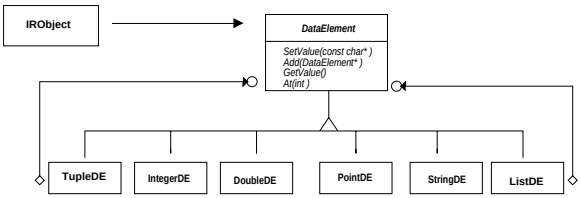

Fig. 4. Data Element Hierarchy

`DoubleDE`. `TupleDE` and `ListDE` are two concrete composite classes derived from the class `DataElement`. TupleDE class is used to hold a sequence of data types and ListDE is used to group a list of same types of DataElement. This type hierarchy is extensible so that new types can be created and added to the hierarchy if needed.

All documents modelled using this data modelling technique are wrapped in instances of a class called IRObject. It has an instance variable to hold a DataElement or any of its derived classes. By this mechanism any complex object of any structure can be represented in an `IRObject`. A collection of documents is grouped in a class called `IRObjectSet`. `IRObjectSet` holds a list of IRObjects. It has functions to return an IRObject given its identifier. An `IRObjectSet` groups same types of documents and heterogeneous types of documents are grouped in different `IRObjectSet`s.

One of the recurrent problems in IR system development is the manipulation of input files marked up in any scheme. We developed a general purpose parser generator mechanism, the details of which are described in Hendry [6, pages. 188-195]. An experimenter can specify the structure of a document in a simple scripting language. The scripting language has constructs to represent atomic components and also complex components. From these scripts, a parser is generated automatically. Details of this scheme is available in [7].

6 Feature Indexing and Representation

In the FLAIR system indexing is initiated by an external specification of the indexing scheme. In this specification, one can denote the index structure name to be used, combinations of filters and converters to be applied and the components of the document on which indexing is performed. The distillation process (as shown in Figure 5) starts with an activity of extraction. An Extractor visits all the documents and extracts all those components needed to build different indexes. Each extracted component will go through a chain of filters and then through a converter before being transformed and placed into an index structure. In the following subsections, this distillation process is explained in detail.

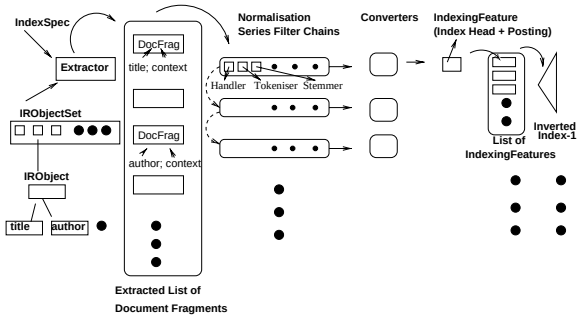

Fig. 5. Distillation Process in FLAIR

6.1 Extractor

The role of an Extractor is to extract different components of IRObjects for deriving indexing features. Extractor takes an IRObjectSet together with a list of component identifiers and extracts the components corresponding to those identifiers from the IRObjects. This list of component identifiers is specified in the index specification. In deriving indexing features from an IRObject, one may need to have the context such as document number, component information etc. Hence, along with the components, the Extractor generates the necessary contextual information. A context and the corresponding data component together are wrapped in an instance of a class called `DocumentFragment`. An Extractor generates a list of DocumentFragments, which can then be used to build one or more access structures. As shown in Figure 5, Extractor extracts document components for all the indexes in one go and stores them in a list for normalisation.

6.2 Filters

Normalisation is achieved in the FLAIR architecture by defining a set of filters and passing the document fragments through this filter set. A filter takes an extracted component (i.e. DocumentFragment) and applies its filtering algorithm to it. In the course of this process it may generate a set of fragments of the document, as in the case of tokenising a piece of text. Currently, we have the following filters built into the library: `Handler, Tokeniser, Stopper, Stemmer`. The filter hierarchy is shown in Figure 6.

In the front of the filter set is the filter Handler which is used to check whether a document fragment can be filtered using the filter chain or not. Tokeniser tokenises a piece of text, Stopper removes stop words and Stemmer applies a stemming algorithm to the document fragment.

These filters play an important role in the distillation process and are designed to provide maximum flexibility and extensibility in deriving indexing features for an IR system. In this distillation scheme, a document fragment passes

through different filters to get normalised before generating indexing features. Depending on the application, a developer has to apply various combinations of normalisation methods, in various orders. There are situations in which one has to apply different normalisations (e.g. phrase indexing and term indexing) to the same component of the document or different normalisation schemes to different components. The combinations of filters and their order is specified in the external index specification thus allowing maximum flexibility in the normalisation. By varying the order of filters in the sequence, we could achieve different forms of normalising. The output of the filtering is a normalised document fragment or fragments. Multiple indexing is achieved in the FLAIR by creating a set of filter chains.

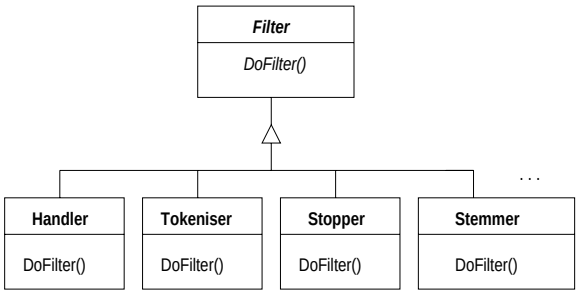

Fig. 6. Filter Hierarchy

A variant of the pattern called 'Chain of Responsibility' [4, p. 223] is used in the design of this filtering mechanism. This filter hierarchy is extensible so that we can create new filters, place them in a filter chain, thus achieving different normalising effects.

6.3 Access Structure

Currently, in FLAIR, we provide abstractions for the inverted index data structure. In an inverted index, features and their corresponding posting list (list of documents in which those features appeared) are stored. The class `InvertedIndex` has functions to provide statistics needed by various retrieval engines.

Building an access structure for an IR system is a complex task, which involves taking a document, generating features from the document and representing these features in a form which is useful for access and involves less storage overhead. These transformations and the resulting representations depends on a number of design parameters involving time-space trade-off. To reduce the size of the database, we need to optimise the size of the inverted index by controlling various parameters like the head of the index, and also the organisation of the posting information. For this purpose, we have defined a set of Converters.

6.4 Converters

Applying converters is the last phase in the distillation process. As shown in Figure 5, the output of the conversion is the indexing features. The idea of a converter is to take a piece of data and convert it into another form. During the

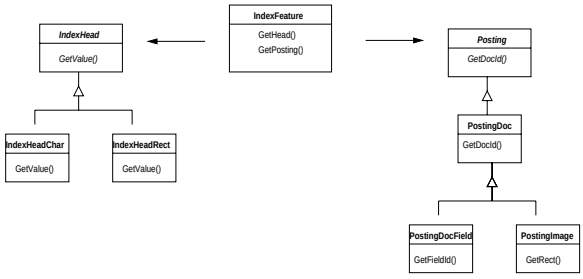

Fig. 7. Two Converter Hierarchies and the Composition of IndexingFeature

indexing process, a piece of document along with its context is converted into an indexing feature. An indexing feature contains two parts: first the head of the index, which goes into the head of an access structure like inverted index; and the second the posting entry which goes into the list part of an inverted index. The classes `IndexHead` and `Posting` represents both the indexing head and posting entry respectively. Depending on the requirement of the application one can specify the type of the IndexHead and the Posting in the external specification of the index. The normalised document fragment is automatically transformed into the specified form. This is achieved by defining two converters in our normalisation scheme. The `IndexHead` and `Posting` are the two converters we use in our index building process. Both the IndexHead and Posting take a DocumentFragment and absorb some or all of the information, thus achieving a conversion. Converters are selected based on an input specification and by extending the converter hierarchy we could derive different type of indexing features.

7 Data Management

In the process of building an IR system, one has to manage various kinds of data like document collection, indexes, and stop word lists. In the FLAIR system the management of these is achieved in a class called `IRBase`. The purpose of an IRBase is to manage all kinds of data arising out of the development of an IR system. These include document collections, index structures built to support different retrieval engines, and collection of stop words. New applications demand management of heterogeneous collections and incorporation of multiple retrieval models in one application. In the FLAIR system, multiple retrieval

models can be built into an IRBase; IRBase has an instance variable called IRModelList to store multiple IRModels. In the IRBase, a set of retrievable objects are grouped in an IRObjectSet. Heterogeneous collections can be organised in an IRBase; the instance variable IRObjectList manages a number of IRObjectSets. Similarly, depending on the application, collection specific stop term set can be maintained in the instance variable StopTermList. IRBase also provides functions for combining results from multiple searches using different IRModels. The relationship between StopTermList, IRObjectList, and IRModelList is not one-to-one; one StopTermSet can be associated with any number of IRObjectSet; one IRObjectSet can be associated with any number of IRModelList.

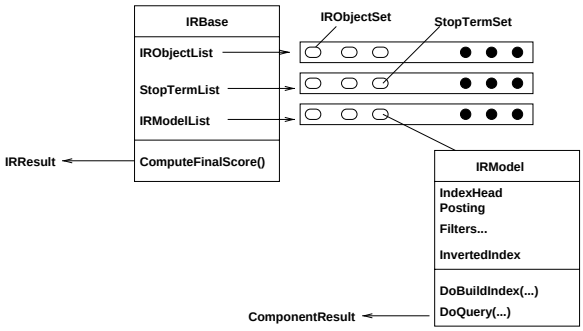

Fig. 8. Data Management in FLAIR

Given a set of IRObjects (IRObjectSet) and an index specification of the system a developer wants to build, FLAIR automatically extracts the appropriate components of the documents, processes them (distillation) and stores the results in access structures (e.g. Inverted Index).

8 Search and Retrieval

The purpose of an IR system is to provide effective and efficient access to the information stored in the system. Once a retrieval system has been built, it can be used for accessing information and a retrieval task involves query specification, query feature extraction, query feature normalisation, matching and result generation. This activity is shown in the right hand side of the Figure 3. A query can be specified in different ways and in general, the representation of the query depends on the retrieval (search) engine (more specifically it depends on the way information is organised in the underlying access structure). Examples of search (query) types include: simple term based search, boolean search, proximity search, structural search etc. In many modern IR applications, one needs to retrieve documents employing multiple retrieval engines, quite often applied to different components of the document. This in turn demands representation of

various types of query mechanisms in one search. FLAIR supports a query specification mechanism that can represent and can be extended to represent various types of query schemes. To achieve consistency the normalisation applied at the building stage of the system is applied to the query feature normalisation.

In FLAIR, a searcher's information need is represented in a class called `IRQuery`. In IRQuery one can model various kinds of queries. IRQuery is modelled using the same data model used to represent the documents. This facilitates the application of the same distillation that is used at the building stage of the system to the IRQuery and hence helps to achieve consistency in generating indexing features. The features resulting from the normalisation are stored in an instance of a class called `QueryRep`. In the case of queries involving multiple components, different QueryReps are used. Finally, the entries in QueryReps are matched with the entries in the corresponding access structure stored in an instance of the class IRModel. The results generated after the matching are stored in an instance of a class called `ComponentResult`. Finally, the the ComponentResults generated are combined together and the final result is stored in an instance of a class called `IRResult`.

The process of computing a similarity value between the query and the documents is organised in an instance of a class called `Matcher`. The `Matcher` class takes the `QueryRep` and an instance of the `InvertedIndex` and generates the results. New matching schemes can be provided by extending the matcher hierarchy.

In the FLAIR system, queries are represented in a simple query description language. In this description, a searcher can specify the name of the `IRModel` to which the query needs to be compared, the confidence of the searcher in that component of the query, and the matching algorithm to be employed. A query parser built into the FLAIR system converts this representation into an instance of the class `IRQuery`. FLAIR employs the corresponding matching scheme from the specification thus allowing us to apply different types of matching schemes to different query components. The inbuilt result combination mechanism uses the confidence values for evidence combination [8]. Developers can provide their own combination scheme by extending the IRBase class.

Developers can specify the kind of matching mechanism they want to apply for a query component in the query description. From this, FLAIR employs the corresponding matching scheme. This allows us to apply different types of matching schemes to different query components.

9 Discussion

A number of object-oriented frameworks or class libraries for IR have been proposed [2, 5, 11, 10]. However, FLAIR addresses many limitations of these approaches. Indeed, FLAIR follows the same approach as ECLAIR [5] by building a framework on top of an OODBMS. However, the similarity ends there, the design and the resulting abstractions are different. No other approaches provides the flexibility of FLAIR in dealing with heterogeneous collections, dealing

with multiple indexes and also specifying and composing systems from external specifications.

The development of the FLAIR system progressed hand in hand with its utilisation for the building of IR applications. We have used this architecture to build a photograph retrieval system [9]. We have also modelled different kind of IR documents (e.g. documents from CISI collection, various TREC collections) using the FLAIR data model.

10 Conclusion

This paper describes FLAIR, an extensible architecture, which can be used to construct IR applications. FLAIR provides abstractions for IR activities so that developers can use them in a mix-and-match fashion. The salient features of the FLAIR library are: creation of representation for documents based on an input description; building multiple access structures based on an external index specification; and query representation from an external query description. Experimenters can change a very large number of parameters involved in a search strategy, for example, what fields in a document need to be indexed, what combination of normalisation has to be applied, etc. One can build new retrieval models and access structures by extending a few of the abstractions provided thus achieving a great deal of extensibility. The FLAIR architecture is flexible in that these building blocks can be combined in various ways at run time to build various retrieval engines without extending any of the classes and without re-compilation.

Acknowledgements

We would like to thank Jan-Jaap IJdens for being a constant source of ideas and criticism. The work presented here has been supported by the Principal's Research Fund at the Robert Gordon University.

References

1. Carroll, J. M., Mack, R. L., Robertson, S. P., and Rosson, M. B. Binding objects to scenarios of use. *International Journal of Human-Computer Studies 41* (1994), 243–276.
2. Cutting, D., Pedersen, J., and Halvorsen, P. An object-oriented architecture for text retrieval. In *Proceedings of RIAO'91, Intelligent Text and Image Handling* (Barcelona, Spain, 1991), CID, France, pp. 285–298,.
3. Frakes, B. W., and Baeza-Yates, R., Eds. *Information Retrieval Data Structures and Algorithms.* Prentice Hall, Englewood Cliffs, New Jersey, 1992.
4. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.
5. Harper, D. J., and Walker, A. D. M. ECLAIR: an extensible class library for information retrieval. *The Computer Journal 35*, 3 (June 1992), 256–267.

6. Hendry, D. G. *Extensible Information-Seeking Environments.* PhD thesis, The Robert Gordon University, Aberdeen, September 1996.
7. Jose, J. M. *An Integrated Approach For Multimedia Information Retrieval.* PhD thesis, The Robert Gordon University, Aberdeen, October 1998.
8. Jose, J. M., and Harper, D. J. A retrieval mechanism for semi-structured photographic collections. In *LNCS 1308 (Proceedings of DEXA 97)* (September 1997), Springer, pp. 276–292.
9. Jose, J. M., and Harper, D. J. Epic: A photograph retrieval system based on evidence combination approach. In *Proceedings of the IFMIP 98 Conference* (May 1998), M. Jamshidi, C. W. Silva, F. Pierrot, M. Fathi, Z. Bien, and M. Kamal, Eds., Alaska, TSI Press.
10. Mills, T., Moody, K., and Rodden, K. Cobra: A new approach to IR system design. In *Proceedings of RIAO'97, Computer-Assisted information searching on Internet* (Montreal, Canada, June 1997), pp. 425–449.
11. Sonnenberger, G., and Frei, H. P. Design of a reusable IR framework. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (July 1995), E. Fox, P. Ingwersen, and R. Fidel, Eds., ACM Press, pp. 49–57.
12. Van Rijsbergen, C. J. *Information Retrieval.* Butterworths, 1979.