# Information Retrieval - from Academic Research to Practical Applications

Hans-Peter Frei
*UBILAB, Union Bank of Switzerland*
*Bahnhofstrasse 45*
*8021 Zurich, Switzerland*
*E-mail: Hans-Peter.Frei@ubs.com*

## Abstract

*Information Retrieval (IR) as a discipline did not originate from an urgent need of professionals to master today's information explosion. Nor is the content of the relatively small test collections employed in the past by IR research in line with the kind of information professionals are faced with at the present time. Only recently IR researchers have started to study real-life problems and have realized that the information which enterprises invariably encounter is highly unstructured, heterogenous, multi- and hypermedia, and of varying quality. Because of the requirements imposed by today's professional life, Information Technology (IT) software becomes increasingly complex. We argue in this paper that it is an advantage to use a framework approach—an advantage not only for operational IR applications but also for research purposes, particularly for the evaluation of indexing and retrieval techniques. The IR framework FIRE is being developed along these lines. The rationale for developing FIRE is discussed, and the structure and some of the basic properties of FIRE are explained.*

## 1 Information and Information Retrieval

It is not the aim of this article to repeat the technological achievements of the past decades nor to predict what technology will bring us in the decades to come. Likewise, the often cited information explosion is a well-known fact and its effect on the life of many professionals should not be reiterated either. Rather, we want to summarize what Information Retrieval (IR) research has accomplished in the past and to what effect the achievements of IR research have been put into practice in real-life environments. Hence, we will reflect upon different kinds of information, upon the value and price of information, and upon the expectations of practitioners concerning IR research results.

It is often argued that information in the past used to consist of text only and, because of this single medium, searching for information was easier. There have, of course, always been other types of information, like speech, gesture, images, etc., but indeed text was the only information type that was stored on a medium (e.g. on paper) and was therefore searchable. With the advent of new recording equipment many other kinds of information started to grow into entire collections. When a need for searching mechanisms emerged, indexing and retrieval methods evolved, often imitating the methods devised for texts. With the extreme mixture of media, called *multimedia* information, the task of retrieving specific information became increasingly difficult. In addition, hyperstructures emerged as in hypertexts or the world-wide web (WWW) which made the searching for specific facts in *hypermedia* collections even more difficult. Solving these problems is the objective of many research projects in the area of IR at the present time.

Value and price of information vary enormously, depending on the demand, the kind of information, its origin, and its supplier. Information can be expensive or free, it can be valuable or useless, it can be accurate or plethoric as we know only too well from the Internet. However, most of the time information has a price and consumers have to decide whether or not they are willing to pay the money asked for. As a consequence, some people pay money for information and others make money by selling information. Information has become a commodity and is easily shipped around the entire world nowadays, in contrast to the information on the outcome of the battle of Marathon for whose deliverance Philippides, the first Marathon runner, paid with his life over two millennia ago.

The emphasis of this paper is neither on telecommunication nor on the price of information. Rather the focus is the discipline of IR. From the term *information* retrieval one could conclude that IR originated from an urgent need of professionals to search for important facts in vast amounts of information. It is remarkable that this is not the case. Rather, IR originated in the field of library science and used to be synonymous with retrieving records whereby the term 'record' signified a library record, i.e. an entry in a library catalog. Partly for this reason the IR research community has had very little impact on those professionals concerned with systematically *collecting, storing,* and *retrieving* information.

These practitioners striving to master today's information explosion often consulted the IR literature in the hope of finding advice. Frequently, they realized after a while that IR research did not concentrate on their problems and often they were frustrated by having understood what IR researchers were suggesting.

Less than a decade ago this situation started to change; IR researchers and practitioners started to interact. But the situation will have to change even more in the not too distant future. It has to change because practitioners want researchers to work on the problems dictated by commercial needs. The hope is that researchers will soon work on these challenging new problems in environments unknown to them before. This change of direction is even accelerated by the increasing policy of granting agencies to require that proposed projects have a high degree of practicability.

In the remainder of this paper we will concentrate on the practitioner's view and requirements as well as on our view of the problem and our efforts to solve it.

# 2 Academic Research and the Practitioner's View

As pointed out above, IR research has for quite some time been moving away from its realm of bibliographic references and lusterless test collections. A good example are the TREC experiments and conferences that started in 1992 as contests between IR researchers

[6]. One started to compare IR system results on a new very large test collection, called TIPSTER, whose size of several GByte exceeds the traditional IR test collections by a few orders of magnitude. In addition, TREC recently introduced non-English documents and included them in the contest.

For years on end, academic researchers studied how to index, store, and retrieve bibliographic references, calling their discipline *information* retrieval rather than *reference* retrieval. Thus, for a long time, IR was concerned with finding a very restricted kind of information and the term 'information' retrieval was a real misnomer. Retrieving relevant bibliographic references is certainly a valid problem useful to some people. But it clearly does not reflect the majority of the problems that have to be solved facing today's information explosion. Business analysts, journalists, and scientists hardly ever need bibliographic references for their work. Most of the time they need facts, i.e. direct information about the problem area they are working in; oftentimes they have neither the interest nor the time to follow-up references, get articles from the library, and read papers.

Yet, it is not only the amount of information that constitutes a problem. While we know reasonably well how to index, store, and retrieve text, non-textual information becomes more and more important. Hypertext and multimedia IR have become buzzwords in the past few years and many an IR researcher has started to extend traditional IR methods to be applied to hypermedia information. Hard problems have been tackled, such as the retrieval of content-related information from collections of video tapes or hypertexts. The simple solution of describing video material or images by textual descriptors [10] is cumbersome and does not work sufficiently well. Determining scene changes in movies is still a research topic, the combining of indexing features from the image and sound track of a movie is another problem of investigation, and the combining of retrieval status values (RSV) across media boundaries is not yet solved. Furthermore, augmenting retrieval algorithms by making them follow hypertext links is a rather precarious undertaking and still needs a fair bit of investigation [4].

It is certainly worthwhile to study all these largely unsolved problems and when they are once solved, the ability to seek information in hypermedia collections will be useful for professionals in the film and TV business as well as for the general public interested in information that happens to be in a hypermedia form. This concerns not only video material, which is growing exponentially, but also an encyclopedia because it contains references to other entries and hence constitutes a hypertext. It is to be expected that the number of widely used operational hypertext and hypermedia systems will grow and it is obvious that this is the case with the information in the WWW. For this reason, the interest in hypertext and hypermedia information retrieval is rising significantly. Most likely, it will boost further research activities in this area.

# 3 Needs of Professional Users

There is a vast literature on the needs of users mostly concerned with how IR systems are interfaced to users, i.e. with human-computer interaction (HCI) problems. HCI is a crucial issue, especially for IR systems that are typically used by a great deal of casual users. Nevertheless, HCI will not be the emphasis of the remainder of this paper.

Rather, we will be concerned with the word 'professional' used in the above title. When referring to *professional* users we are not saying that we consider some users professional and others unprofessional. However, we explored the IR needs of users working for a large financial institution and called their needs 'professional'. These needs contrasted sharply to the more fictitious needs of 'laboratory' users who participate in IR experiments. We examined both the vast amount and the various types of information these professionals have to deal with. Not unexpectedly, we still found a great deal of text on paper. In addition, text in machine-readable form plays an ever-increasing role: text in word processor formats, text delivered by news feeds, spread sheet data, etc.

In addition, we found important non-textual information, such as tables (balance sheets are basically tables), graphs, charts (e.g. organizational charts), numeric data, etc. The different types of information are usually mixed: a table contains figures and text, an overhead transparency may contain text, figures, charts, and even images. Also the sources are manifold, a great deal of information comes from internal sources and is, therefore, easy to monitor and control. On the other hand, there is also external information from magazines, newspapers, public relations statements, electronic news wires, and the like.

In comparison to other information-gathering activities, a different way of classifying pieces of information became apparent: the *quality of information*. The professionals we talked to deal with rumors, information from mass media, from reputable journals, from official sources, and from internal sources. The quality of information depends on various factors and, naturally, the value and price of information vary considerably as was already mentioned at the outset.

In short: our professionals are confronted with *unstructured*, *heterogeneous*, *multimedia* information of *different quality*. Their problem is to find specific facts about well-defined business sectors, over a given period of time and possibly restricted to a given geographical area. In most cases both internal and external information has to be taken into account. In IR terms and with the presently available techniques this means that a series of queries has to be performed on large multimedia collections. Each single query usually depends on the result of the previous one and they form an entire iteration of queries.

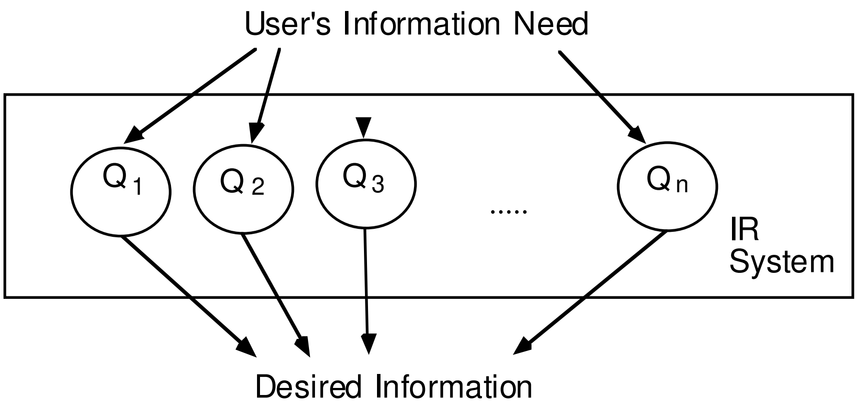


Fig. 1: User submits sequence of small queries

User interfaces for supporting this complex task must allow to keep track of navigational moves, partial results, and the query history. Such interfaces are emerging at the present time [7]. They support the extraction of useful information from individual query results and help in assembling partial results. The sought information still consists in part of many pieces of semi-relevant information that were put together by the user (Fig. 1)

As mentioned before, the needs of a library user looking for a book on fly-fishing are relatively simple compared to the needs of a professional confronted with today's information jungle. A professional user needs extremely powerful evaluation procedures to capture real problems, procedures that return *facts* rather than numerous partial query results that have to be combined manually. Ideally, such procedures must accept *inquiries* rather than queries and should be able to generate automatically a series of simple queries where needed. Likewise, the results of simple queries should automatically be assembled and turned into useful facts (Fig. 2).

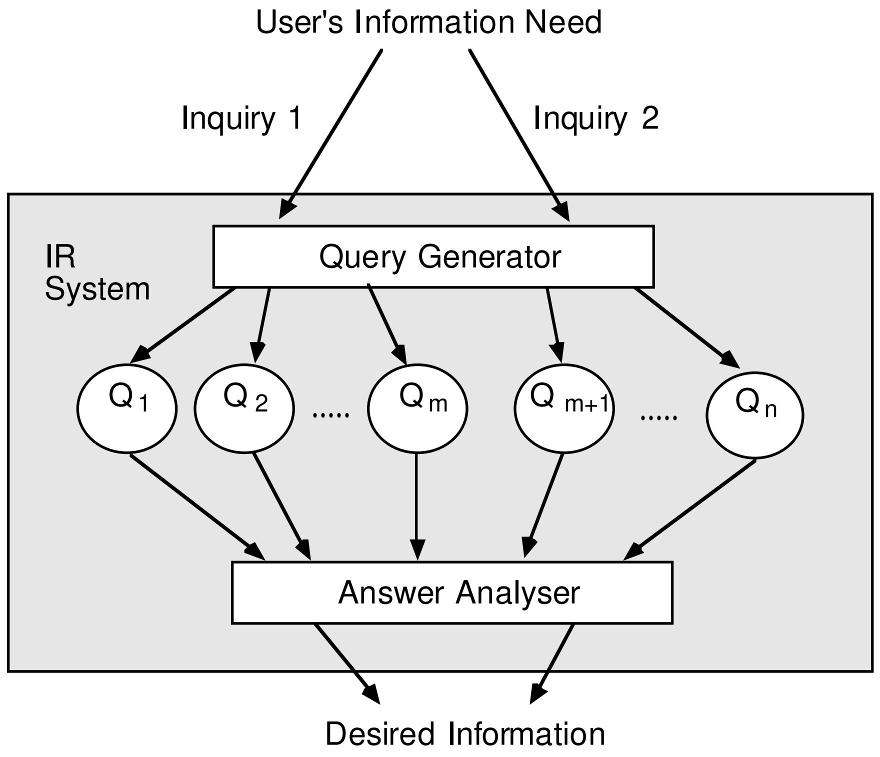


Fig.2: User submits inquiries

The IR system of the future should therefore accept real user inquiries and return answers that help these users to master today's information flood. Only in this way is it assured that the value of information contributes optimally to the problem-solving process.

# 4 Quest for a Flexible IR Framework

From the needs pointed out above it becomes clear that we not only need more complex systems but also a great deal of flexibility since it is difficult to devise an IR system that will adhere to requirements not yet fully known at the time of its design. Implementing a new IR system whenever the requirements change is neither practical nor economical. Similarly, it is unfeasible to extend an existing traditional system to adhere to new requirements due to its inappropriate architecture and missing tools.

In other contexts, these problems have been solved by putting together subroutine or code libraries that can be used by applications. Consider numerical or statistical analysis [1] where a number of useful subroutine libraries emerged a long time ago and are now in general use, such as NAG and IMSL. The goal of such libraries is to define, implement, and put at the users' disposal collections of pieces of code, each of which constitutes the implementation of a specific function. The functionality and interfaces of such subroutines or procedures must be exactly defined and carefully documented for the application programmers. Subroutine libraries have the advantage that proven algorithms can be used and well-established code becomes part of the applications. The programmer just has to choose the appropriate procedures—which is not trivial in all cases—but has neither to read nor to fully understand the code that is invoked. It goes without saying that the functionality and the interfaces have to be fully understood.

An alternative to subroutine libraries are *frameworks*. General application frameworks, such as ET++ [12] or the Microsoft Foundation Class Library [9], are emerging at the present time. In contrast to a code library, a framework is a program skeleton defining the basic concepts of an application domain. This includes the definition of how individual components cooperate. In other words, a framework not only contains specific algorithms and their implementations, but it also defines the entire application architecture. Therefore, a framework constitutes the major part of an application which can still be flexibly adapted to specific requirements. In contrast, the programmer employing a code library is confronted with a take-it-or-leave-it scenario: a subroutine can only be used if it has the exact functionality required and, in addition, an interface that fits into the application under development. Neither the functionality nor the interface can normally be adapted by the application programmer. This is different when a framework with suitable generalizations and abstractions is available. It can flexibly be adapted to the needs of the application and therefore constitutes a much more flexible boundary to build on.

IR systems are usually not structured in such a way that they can easily be adapted to the specific needs of a new application. This also holds for systems like INQUERY [3] or SMART [2] whose code has occasionally been reused by others than their developers. In most of these cases, the systems have been used for application areas similar to the ones they were designed for. Therefore, either they could be used as they were or only small adaptations were necessary. A more flexible approach was pursued with the object-oriented class library ECLAIR [5] providing algorithms for automatic indexing of texts and best-match retrieval. The goal of ECLAIR as a class library was mainly to provide IR functionality in the form of implemented, reusable classes. The emphasis of a real framework approach, on the other hand, is to provide an application skeleton that can be extended to suit specific situations mainly by exploiting the inheritance properties of objects.

Another reason for adopting the framework approach in IR is that there is only little consensus as to which functions have to be provided, which algorithms have to be used, and in which way they have to be implemented in order to be of advantage to different IR systems. As long as only single terms are derived from unstructured ASCII texts, one could certainly find a simple way of providing an appropriate indexing function. However, as pointed out in section 3, different kinds of input have to be considered, such as structured texts, tables, spreadsheets, and the like.

Finally, the framework approach is perfectly suited for a research situation where different algorithms have to be developed and compared with each other. Classes of a framework can easily be instantiated in turn or can even be exchanged, an ideal situation for experimentation.

In cases such as the ones described above, the subroutine library approach becomes far too restrictive and rather unsuited for the development of a flexible system. The IR system we envisage should not only be adaptable to specific application areas but also to requirements that may arise after the system has been deployed.

# 5 Design of an IR Framework, the FIRE Approach

The "Framework for Information Retrieval Applications", called FIRE, is designed as a flexible and extensible framework for developing a wide range of specific IR applications [11]. Among others, it has to provide a great deal of basic IR functionality for various media and flexible modeling mechanisms suited to different document types and structures. One of the problems to overcome is that it is largely unknown in advance which kind of documents have to be managed in a specific application. In addition, we do not know how these documents will be represented and what functionality will be required. Thus, a useful IR framework must provide options for applying various indexing and retrieval models and methods. Most importantly, the framework must allow to add new functionality when necessary for specific applications.

Developing an application becomes chiefly a matter of extending and specializing the framework in the direction of the application specification. This is done mainly by filling in gaps or by providing application-specific functionality. In other words, instead of programming components, the application developer largely selects and plugs together functionality that is already part of the framework. A substantial difference to invoking existing components from an emerging application is that a framework provides both the thread of control and the architecture of the application.

FIRE is implemented in C++ using ETOS, an integration of the ET++ application framework [12] and the object-oriented database system ObjectStore [8]. In other words, FIRE takes advantage of a more general application framework, i.e. the developers of FIRE profit from an architecture and functionality that others devised and implemented. FIRE itself is also such an architecture and plays the same role for developers of IR systems that ET++ plays for the FIRE developers: it shifts substantial parts of the application development task from the developer to the framework. The developer mainly chooses from alternatives that are already supported by the framework and only adds own components when absolutely necessary. Meta information plays a crucial role in representing such choices. Similar to the meta information already provided by ET++, other meta information is supported by FIRE, such as the choice of indexing algorithms and parameters.

When an application is executed, FIRE inspects meta information parameters to decide which functionality has to be invoked. It is the task of the application developer to specify values for these parameters. Default parameters are always available and can be overridden at the framework, application, and user level.

Another important property of the FIRE design is the emphasis on extendibility, as it is largely unknown what kind of functionality will be needed in a multi-media IR environment. It must be possible to add both new media types and new functionality without interfering with existing parts of the framework or with existing applications. If the core framework only works on the interfaces of abstract classes, new concrete subclasses can be added without changing the core framework.

The software architecture of a typical application developed with FIRE is depicted in Fig. 3. Application-specific code fills gaps in the framework or extends it. The underlying integration of ET++ and ObjectStore also provides functionality to FIRE that the developer can profit from.

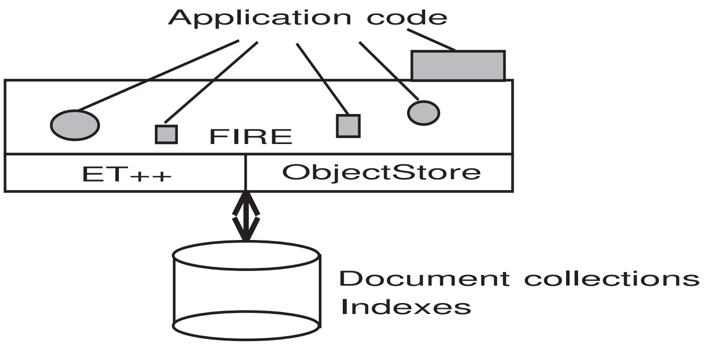


Fig. 3: Architecture of a FIRE application

# 6 FIRE Architecture

## 6.1 A Glance at the FIRE Class Hierarchy

FIRE basically consists of a single-rooted object-oriented class hierarchy that is both large and complex. For its sheer size and complexity it cannot be fully presented in the space available here. Therefore, this chapter will solely convey a feeling for the structure of the class hierarchy, explain how IR components are embedded, and give the rationale for choosing this specific design.

Fig. 4 shows the class *InformationObject* that constitutes the root of the hierarchy as well as a few of the most important subclasses. The root defines basic operations for managing and manipulating information units, such as *create*, *remove*, *present*, *edit*, etc. These operations are abstract operations whose actual implementation is defined in concrete subclasses [11].

The subclasses of *InformationObject* such as *ReprInfoUnit* (document), *InfoObjectElement* (data element), and *Index* contain further subclasses that model specific IR components, in particular data and methods. Also the other classes of the framework represent various forms of meta information and functionality for supporting entire applications. As was explained in section 4, this support of entire applications is what mainly distinguishes a framework from a code library. FIRE provides the *outline* of an IR application whereas a code library would only supply subroutines which can be invoked by the application that has to be designed and written from scratch.

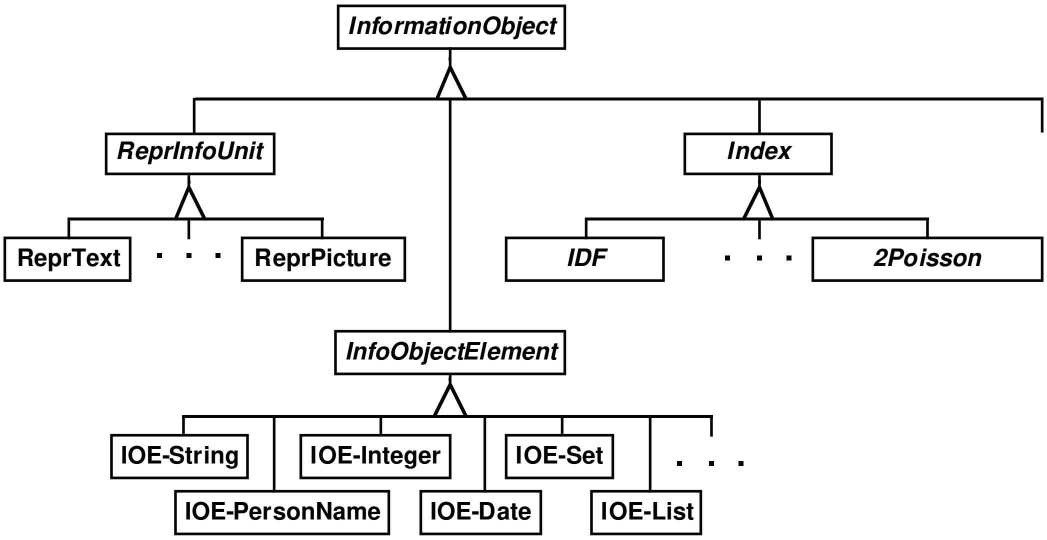


Fig. 4: Top part of the FIRE hierarchy

Consider the abstract class *ReprInfoUnit*. It controls the modeling of documents; its concrete subclasses define how certain information types needed by an application are represented. If the type is text, the application developer first checks the framework for an appropriate subclass for text. Since there is a class *ReprText*, the question is if its features such as *Title*, *Author*, *PublicationDate*, etc., are suited to the application under development. If they are not, a new text class would be devised which is then used—instead of *ReprText*—by the particular application. Note that this new text class then forms part of the framework and can be used by applications that are developed at a later point in time.

The features of *ReprText* in turn are taken from a collection of predefined data types represented by another abstract class, in our case the class *InfoObjectElement*. At first glance, this decoupling of document types (*ReprInfoUnit*) and data types (*InfoObjectElement*) looks complicated. The reason is, however, a gain in flexibility. The subclasses of *ReprInfoUnit* are relatively simple whereas the subclasses of *InfoObjectElement* represent media formats and can be quite elaborate. On the other hand, they represent the basic building blocks for all kinds of complicated documents and can, therefore, be reused in many contexts.

New information units may become necessary even after the deployment of an IR system; they can easily be added as (relatively simple) subclasses to *ReprInfoUnit*. The (relatively complicated) media formats have to be implemented just once whereafter the corresponding subclasses of *InfoObjectElement* can be used over and over again by subclasses of *ReprInfoUnit*.

For every data type described in *InfoObjectElement* the framework contains specialized IR functionality, such as tokenizing, indexing, matching, etc. If there are no pictures in the data collection, there are no instances of the subclass *ReprPicture*. Likewise, if the data collection would contain speech and speech were not supported by the framework, a subclass *ReprSpeech* would have to be added by the application developer. In this specific case, also *InfoObjectElement* would presumably not contain speech-specific basic data types, so that a few additional subclasses would have to be developed. Such newly added subclasses are automatically available to every user of the framework just as any other framework component. Most importantly, additions require no changes to the framework core nor to any of the existing applications.

Therefore, a specific IR application can be developed by choosing and plugging together framework components. In case there is no component for a data type or no functionality necessary for the application under development, some of the following actions have to be taken:

- The needed document type does not exist: a new *ReprInfoUnit* subclass has to be implemented, an easy job when the necessary data types are available as *InfoObjectElement* subclasses.
- The needed media component does not exist: a new *InfoObjectElement* subclass has to be implemented; depending on the complexitiy of the particular media, this can be a rather demanding job.
- The desired weighting method does not exist: a concrete subclass of the abstract class *Index* has to be developed.

## 6.2 Indexing and Weighting

Like many other classes belonging to the same level of the class hierarchy, also *Index* is an abstract class. It does not provide specific IR functionality but solves general subtasks necessary to manage indexing features and to compute RSVs.

Consider the method *addIFs* belonging to *Index*. It adds a set of indexing features to an existing index. When doing so, this method also checks whether the feature to be added is compatible with the already existing index, i.e. if the same indexing method was applied to the newly arriving feature as was applied to the old ones. Another method of *Index* is called *update*. It finally inserts the previously added indexing features into the index. The two processes of adding and updating are separated because the actual insertion of indexing features into an existing index can cause a great deal of computing and re-sorting of the existing index.

Another method called *retrieve* serves for retrieving information units by evaluating single feature queries. This is again a subtask, since single feature queries usually occur as parts of more complex queries. Such a compound query is decomposed into single feature queries and the results of these single conditions are combined by a *ReprInfoUnit* object that computes the similarity between two information units, e.g. between a document and a query.

Concrete subclasses of *Index* are responsible for assigning weights to features. Each subclass supports a particular weighting scheme, such as Inverse Document Frequency (IDF), discrimination weighting, etc. Note that these subclasses are implemented such that the weighting schemes can be applied to any kind of feature. In other words, the same subclass *IDF* can be applied to stemmed keywords, to phonemes, to graphical features, and the like. Hence, an IR application developer needs only to devise a new subclass of *Index* when an entirely new weighting scheme is to be supported.

### 6.3 Present and Future Developments

Up until now the implementation focus was clearly on the infrastructure of the FIRE framework which means that most of the basic information types are implemented, such as *string*, *text*, *integer*, *date*, *personname*, *speech*, etc. In addition, a variety of stemming and matching algorithms exist that support English and German text and speech. In the indexing part, simple hashing is supported that has been taken from ET++. The goal is to use the ObjectStore indexes directly to increase efficiency.

Fireworks/SketchTrieve is a user interface to FIRE that was developed by our project partner from Robert Gordon University, Aberdeen [7]. Another user interface also running at the present time is a WWW/Netscape interface.

At this time, two small test applications are running on top of FIRE. One is an application that handles the publicly available HCI bibliography of Ohio State University and ACM SIGCHI, the other handles a set of about 400 speech documents of BBC Radio 4 news broadcasts.

The FIRE system so far consists of roughly 200 classes, about 100 of them contributed by Fireworks/SketchTrieve. This amounts to more than 30,000 lines of code altogether.

## 7 Conclusions

Despite the narrow scope of early IR research, a fair number of useful methods and algorithms have been developed mainly for indexing, storing, and retrieving texts. In contrast to former days, information has long lost its automatic association with text. In addition to text there are graphics, image, speech, video, and mixtures of all these information types. These 'new' information types gain significance not only in everyday life but also within the professional realm of enterprises that are faced with today's information flood. As a consequence, the methods and algorithms devised in the early days of IR have to be reconsidered, adapted, or even re-developed.

New IR methods and algorithms are now emerging because of the efforts of both IT professionals in companies and IR researches in academia. The IR systems emerging must have the ability to cope with hypermedia information, e.g. hypertexts and a multitude of different media. The problem in enterprises is often that it is unknown at the time of purchasing or devising new systems what kind of information must be considered and—above all—in what form this information will be available when the IR system is in operation.

For these reasons, we started to develop an IR framework that offers not only basic IR functionality but also the thread of control and the structure of the yet unknown IR application to be devised. This framework, called FIRE, allows a great deal of flexibility to a system developer and provides, at the same time, maximum support.

## Acknowledgments

I would like to thank Gabriele Sonnenberger and Tore Bratvold who designed the FIRE framework in cooperation with the IR group of Robert Gordon University, Aberdeen, Scotland. Many students contributed valuable parts to the implementation of FIRE in the course of internships at UBILAB. Particular thanks go to Tore Bratvold who read earlier versions of this paper and whose comments have greatly improved the quality of the final product.

## References

[1] Boisvert R F, Howe S E, Kahaner D K: GAMS: A Framework for the Management of Scientific Software. *ACM Trans on Math Software*, Vol 11, No 4, 1985, pp 313-355.

[2] Buckley Ch, Salton G, Allan J, Singhal A: Automatic query expansion using SMART: TREC-3. In: Harman D K (ed.): *Proc 3rd Text Retrieval Conf (TREC-3)*, NIST Special Publication 500-225, 1995.

[3] Callan J P, Croft W B, Harding S M: The INQUERY Retrieval System. *Proc 3rd Int Conf on Database and Expert Systems Applications*, 1992, pp 78-83.

[4] Frei H P, Stieger D: The Use of Semantic Links in Hypertext Information Retrieval. *Information Processing & Management,* Vol 31, No. 1, 1995, pp 1-13.

[5] Harper D J, Walker A M: ECLAIR, an Extensible Class Library for Information Retrieval. *The Computer Journal*, Vol 35, No 3, 1992, pp 256-267.

[6] Harman D: The TREC Conferences. *Proc HIM '95 (Hypertext - Information Retrieval - Multimedia)*, UVK Universitätsverlag Konstanz, 1995, pp 9-28.

[7] Hendry D G, Harper D J: Coordinating Information-Seeking on Interactive Displays. In: Collier M and Arnold K: *Proc 2nd Int Conf on Electronic Library and Visual Information Research*. Aslib, London, 1995, pp 127-136.

[8] Lamb Ch, Landis G, Orenstein J, Weinreb D: The ObjectStore Database System. *CACM*, Vol 34, No 10, 1991, pp 50-63.

[9] Microsoft: *Programming with MFC and Win32*. Microsoft Press, Redmond, WA, 1994.

[10] Ogle V E, Stonebraker M: Chabot: Retrieval from a Relational Database of Images. *IEEE Computer*, Vol 28, No 9, September 1995, pp 40-48.

[11] Sonnenberger G, Frei H P: Design of a Reusable IR Framework. *Proc 18th Int ACM SIGIR Conf,* ACM Press, New York, July 1995, pp 49-57.

[12] Weinand A, Gamma E: ET++, a Portable, Homogeneous Class Library and Application Framework. In: Bischofberger W R, Frei H P (eds): *Proc UBILAB Conf '94*, Universitätsverlag, Konstanz, 1994, pp 66-92.