Query Modelling for IR Interface Design

S. Jones, T. Do, M. Hancock-Beaulieu, A. Payne and S. Robertson,
Centre for Interactive Systems Research, Dept. of Information Science,
City University, Northampton Square, London EC1V 0HB
e-mail: sj@is.city.ac.uk

In an interactive IR system which supports various methods of query enhancement, a searcher's initial description of his requirements may consist of a simple word or phrase, which by the end of a search session has grown into a complex structure by the addition of terms from different sources, perhaps weighted according to different criteria, and related to one another in various ways. On the one hand it is necessary to represent and manipulate this information effectively, on the other to hide much of its complexity from the non-expert user. We report on work in progress to define an object-oriented model of a probabilistic IR system (OKAPI), the central component of which is the query itself. We consider how to represent queries both internally and at the user interface level, and their relationship with other components of the model. The model will form the basis of a configurable user interface, which allows controlled experiments to be undertaken, and could be adapted to the needs of different users accessing different databases. Implementation will involve the use of a high-level interpreted scripting language for overall control, communicating with an internal model and an interface model, designed and developed using object-oriented techniques.

INTRODUCTION AND OBJECTIVES

In principle, a probabilistic IR system provides many subtle combinations of useful retrieval strategies; the challenge is to present this increasingly sophisticated functionality in a way which is understandable, giving users some sense of control over their information search. Even more important, the 'division of labour' between searcher and system must be such as to ensure maximum effectiveness overall. Probabilistic IR exploits a great deal of statistical information (e.g. frequency distributions of query terms in a document database), which a human cannot possibly assimilate and use in the same way. Under artificial laboratory conditions[1], completely automatic retrieval methods easily outperform those involving interactive searching. In more normal circumstances it is of course human users who

want the results, and who can make decisions only on the information which is currently before them.

Work is currently in progress to build an easily configurable graphical user interface to OKAPI[2] - a probabilistic search engine which has operated successfully under widely differing circumstances. The intention is to make it possible to vary design parameters about what to show and what to hide, where to operate automatically and where merely to offer suggestions, and, by conducting experiments and logging results, to discover how best to combine the capabilities of a human searcher and a probabilistic system.

One point to investigate is how a *thesaurus* can be called upon to suggest useful new query terms throughout a search session. Earlier experiments[3] have examined the effectiveness of thesaurus-based query enhancement before document retrieval. In that case user-selected thesaurus terms were added to the initial query, allowing the probabilistic ranking system to make more precise discriminations between document weights. The new system will permit the thesaurus to be consulted at every stage of the search, and even take the initiative by identifying appropriate candidate terms for selection.

Another important research objective is to look at strategies for *query expansion via relevance feedback.* It is standard practice in OKAPI to require a YES / NO relevance judgement on all documents seen during a retrieval session, and to expand queries periodically by extracting new terms from relevant documents. A previous project[4] investigated the effect of user intervention in this process, finding that it was not always beneficial. The new system will be used to investigate other ways of presenting and evaluating this option.

OBJECT-ORIENTED DESIGN PRINCIPLES

Given these objectives, our development methods and tools must enable programmers, designers and experimenters to understand and communicate about the system, and, eventually, to define and select different versions of the interface for different experiments. After investigating a number of possibilities, the following strategy has been adopted:

- Build an *underlying model* of a retrieval system using an object-oriented programming language (C++). Components of this model include familiar 'objects' such as terms, queries, documents, hit-lists, etc.
- Build an *interface model* containing corresponding display objects, and also control objects like buttons to initiate searching or query expansion.
- Use an *interpreted scripting language* to communicate with the compiled program, setting and interrogating object properties, and exercising overall control of system behaviour.

Object-oriented principles have been used elsewhere in the design of general-purpose IR systems. Many of the objects which form part of our model have counterparts, for example, in the Eclair system[5]. For us, a major advantage of the approach is that it distinguishes the three design levels of **model**, **view**, and **control**, introduced by the early proponents of the O-O paradigm at Xerox Parc[6].

The underlying model is described in terms of *objects*, each comprising a collection of *properties* (data values) and a set of *methods* to operate on those properties. Defining an object involves defining what *messages* it accepts and how it responds: for example a 'document-set' object might respond to messages to insert, delete or display a member of the set.

Many different *views* of the underlying model can be shown, containing more or less detail as required. For example a basic 'term' object has certain properties (like source and weight), which may or may not be seen when that term is displayed as a component of the current query. The same applies to document weights in a hit-list. Different views of a basic object can be implemented by defining different interface objects to contain them. Interface objects are themselves defined in terms of lower-level widgets for an X-Windows environment, though they could equally well be mapped onto similar mechanisms for PC-Windows.

Handling the *control* functions within an event-driven system is acknowledged to be more complex than it was in earlier 'modal' environments where users were presented with linear sequences of choices and actions. However, it provides better potential for successful interactive retrieval which, as already suggested, demands close co-operation between searcher and system. Our objective is an interface which gives users the feeling of being in charge, but which throughout the search session is

presenting *suggestions* - about possible new terms to include or options to explore - in a helpful but not too obtrusive manner.

Even in a non-modal system, however, not all possible actions can be carried out at all times: we cannot perform a search before entering a query, or delete or insert a non-existent term. Interface control objects such as buttons and menu options should appear only when they are applicable, and the underlying control of *that* process must involve, somewhere, the use of standard program control structures, particularly conditionals. Furthermore, system behaviour may vary according to the version of the interface required for particular experiments: for instance whether the system performs fully-automatic query expansion after relevance feedback or allows the user to intervene by selecting or rejecting system-extracted terms. In the interests of flexibility, designers and experimenters use the interpreted scripting language TCL[7] to specify the features of interface objects and handle overall flow of control.

SYSTEM OVERVIEW

Figure 1 shows the relationship between the major system components. We are working towards a distributed system in which these components may run on different machines, and communicate with one another over the network.

The most important is the OKAPI **Basic Search System** (hereafter known as the BSS). The BSS supports a low-level query language based on ISO SR (Z39/50), but with additional very important functions to handle term stemming, probabilistic term weighting, document ranking, and the extraction and re-weighting of terms from relevant documents. From our point of view this search engine is a black box - the object which represents it in our model simply generates and receives the relevant messages.

In principle, a **document database** can be considered as a separate object, which stores and returns information about document structures and database statistics, and, of course, responds to requests for particular identified documents. In practice this function is also at present performed by the BSS, though further logical separation may be necessary when the interface is required to deal with several databases simultaneously.

The **thesaurus** is already a separate entity, managed by a standard relational DBMS. For earlier experiments the thesurus had its own interface, based on Oracle Forms; for the new integrated system it is accessed over the network from the C++ program using embedded SQL. The object which represents it in our model handles two basic messages:

- given an initial query, find the corresponding 'best-match' terms in the thesaurus,
- given a term, find its immediate connections, i.e. broader, narrow and related terms.

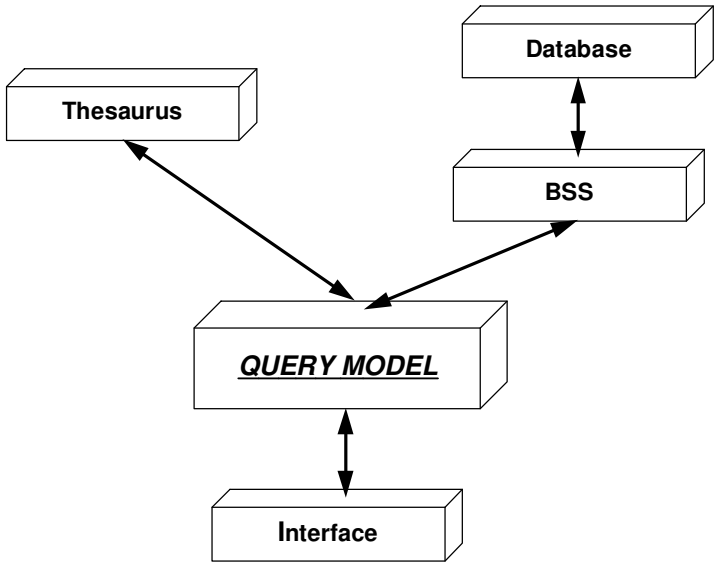

Fig 1: System Components

THE QUERY MODEL

At the heart of the model is the query itself: an expression of a user's information need which, in an interactive environment, develops during a search session by exchange of ideas between system and searcher. In

principle, every relevance judgement on a document makes a potential (perhaps very subtle) change to the current query, and every change to the query makes a potential change to the currently-retrieved document set. Our concern here is how to represent and manipulate this structure within the environment described earlier, and how to define and store successive 'states' of the query so that it is possible to return to an earlier stage in the search session.

Figure 2 gives an overview of the query model at the time of writing. though some further refinements are likely in future. One basic decision is to characterise the representation of a query at two levels: one containing information directly relevant to communication with the BSS, the other requiring a different sort of manipulation either before or after a probabilistic search. The reason for this split is that we may wish to deal with linguistic, pragmatic, or subjective aspects of the query for which at present we have no ready interpretation within the scope of a 'pure' probabilistic system. However the probabilistic system will in fact do most of the useful work, so that level is considered first and in most detail.

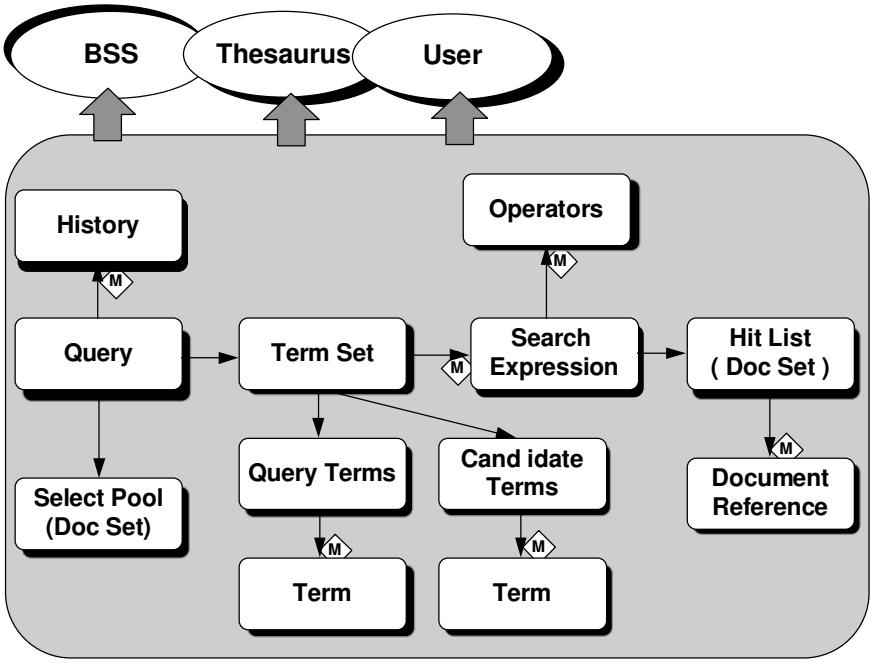

Fig 2: Overview of the Query Model

The Probabilistic Level

The fundamental component of a query is of course the **term**, which in its simplest form consists of a single word. It is traditional in IR to treat short sequences of words (phrases) as composite terms, particularly where they are part of a controlled vocabulary derived from a thesaurus, and assigned as document descriptors. This is certainly the case with OKAPI, where such descriptors are identified once and for all at the indexing stage. (The Eclair system[5] likewise defines a term object as a character string which is stored in an index .)

For the Enquire interface we expect the thesaurus to be the main source of composite terms, which will be displayed to the user as candidates for inclusion in the query. They will be identified in the following ways:

- By an initial 'partial match' with the original query,
- By user-directed thesaurus navigation, to find broader, narrower and related terms,
- By extraction of descriptors from documents judged relevant by users.

At the same time it may be necessary to deal with *user-defined* phrases, composite terms which are not pre-indexed. This is handled by assigning to such terms an *adjacency* property, which is used to select an appropriate BSS operator. There are of course many possible forms of the adjacency / proximity operator, allowing tests for co-occurrence within various word-spans and grammatical units, and in the interests of generality our model allows for any such operator to create a composite term. Actually searching with composite terms involves using adjacency in combination with other operators - a task performed by the *search-expression* object described later in this section.

A term object is represented by both its original inflected form and its *stem*: the character-string by which it is indexed. Every query term is assigned a *weight,* calculated initially on the basis of inverse frequency in the database, but later taking into account its frequency of occurrence in relevant documents. Information is also retained about its *source*: was it entered by the user, selected from the thesaurus, or extracted from a set of

relevant documents? In principle this property could be used by a weighting algorithm, e.g. user-generated terms might be considered more important than those derived by the system. It will also affect the way in which the term is displayed in an interface object.

Individual term objects are held in a **term-set.** This object contains one sub-set of terms comprising the current query, and one sub-set of system-derived candidate terms (following thesaurus lookup and / or term extraction) which are available for addition to the query. Typical methods associated with a term-set are those to *insert* new candidate terms, and to *move* terms from the candidate sub-set to the query sub-set (or vice-versa).

A term-set may be used in one or more **search-expressions**, the detailed construction of which will vary with the experimental conditions. Methods implemented with objects of this type will select or reject terms according to given criteria, and combine them with one or more **operators** in calls to the BSS. In the most general case, a search expression could be a *recursive* structure built of operators, terms, and lower-level search expressions.

Encapsulating the construction and representation of BSS messages in this way should provide the necessary flexibility for testing different search strategies and retrieval algorithms. Earlier experiments[3] compared the performance of an original query as formulated by the user with various 'enhanced' versions containing additional thesaurus terms. In the system now being developed, such comparisons can be made by defining search-expression objects with identical interfaces but different internal behaviour. For controlled experiments, two or more such objects may be called upon with the same term-set during the same search session, and their results combined before presentation to the user for judgement.

As already indicated, the BSS actually carries out the work specified by search expression operators, but for completeness they are defined as objects in the current query model. They fall into three sub-classes: *best-match, boolean* and *adjacency*. We intend not to support explicit reference to boolean operations for the naive user interface currently under construction, though something of their effect could be achieved via the 'filtering' mechanism described in the following section. So best-match will be the default: the particular version being selected perhaps as an experimental variable and defined as a search-expression method.

A best-match operator accepts a set of weighted terms and / or search expressions, and produces a new search-expression with a single composite weight. Given the same input and a particular document, it will also yield a weight reflecting that document's probable relevance to the search-expression. Boolean and adjacency operators identify only sets of unweighted (and therefore unranked) documents; however a composite weight for such sets can be calculated if they are used in a subsequent best-match operation. In complex query expressions containing operators from different sub-classes, it will be necessary to define precedence rules: for example any adjacency operations must be performed *before* the more general best-match or boolean operations.

Any search expression has a corresponding hit-list, an instance of the object class **document-set**. Elements of a document set are **document references**, comprising a unique *document identifier* and, for sets generated by a best-match operation, a *weight*, i.e. an accumulated score of constituent term weights which is used as the basis for ranking. Following examination by the user, the document will also have a *relevance judgement*. Relevant documents identified during a search session will be members of another document set - the pool - which is used by the system for term extraction and query expansion. Methods supported for these objects will include those to:

- move a document reference from the hit-list to the pool,
- compare a new hit-list with an old hit-list to show or record the effect of changing a query,
- merge hit-lists generated by different search expressions for experimental purposes,
- use one hit-list as a 'filter' for another - see the discussion below on query facets.

We can now suggest that the 'state' of a query at any point in a retrieval session is defined by: the pool of relevant documents, the current term-set and search expressions, and the corresponding ranked hit-list. All these objects could be saved and restored in order to support a *history* function; our intention is to save all but the hit-list immediately after a search has been performed, and restore the earlier state by reading this information back and repeating the search.

The Non-Probabilistic Level

There is in fact no such thing as a *pure* probabilistic IR system. All systems are hybrid to some degree; this is demanded and encouraged by the very practical nature of the IR task. The materials with which an IR system operates - documents, queries, thesauri - have linguistic, psychological, and cognitive properties which ought to be taken into account by models and theories about retrieval. In practice, many ad hoc features are now incorporated into working systems to improve their performance, or at least users' perceptions of it. Keen[8] gives an interesting selection of examples: so-called 'devices' which can be provided for an expert user to adjust the effect of a ranked search. They include such things as query phrase marking (adjacency), MUST obligatory term flag and MINUS unwanted term flag (equivalent to boolean AND and NOT), field weights and searcher weights (allowing users to specify the subjective importance of particular terms occurring in particular places). Our own discussions have identified two other issues to consider:

Thesaurus term relationships: At its simplest, a thesaurus is simply a pool of potentially useful search terms for users to select, but in the thesaurus itself those terms are linked, directly or indirectly, by hierarchical and associative relationships. We may wish to retain information about relationships once the terms have been added to the query: at the interface so as to help the user conceptualise his query more satisfactorily, and in the search so as to make finer distinctions between matching documents. For example, where both a broader and a narrower descriptor term occur in a document, it can be argued that the broader term is redundant and should be ignored, whereas if the narrower term is not present, the broader term should somehow be treated as a weaker substitute for it. Clearly this has implications for the query model defined earlier, in that a more complex structure must be imposed on the simple term set, and for the construction of search expressions, in that the thesaural term relationships will need special interpretation in the context of a weighted search.

Query faceting: Some queries clearly involve a combination of two or more topics or concepts. In the query `air traffic control expert systems artificial intelligence`, for example,

we might expect that documents of interest will be those which discuss the application of expert systems or artificial intelligence techniques to the business of air traffic control. We refer to these topics as different *facets* of the query, although aware that this use of the word is at the looser end of the spectrum in traditional IR terms. In a boolean query language, separate facets can be explicitly combined with the AND operator; in a probabilistic system it is desirable to ensure that each is given its due weight regardless of variations in individual term or word frequency.

Earlier experiments with query faceting as a preliminary to thesaurus search[9] suggest that some users do wish to partition their queries in this way and to collect sets of thesaurus terms representing each individual topic. This facility will be retained in the new interface, and extended by enabling users to pursue separate document searches for each topic during a single retrieval session. One consequence for our model is that several discrete term-sets may need to be stored simultaneously, and at the interface an easy way of switching between them supported.

Eventually however the separate facets must be brought together to identify the documents of principle interest. One approach used in recent experiments was to generate a query expression in which all terms from the same facet were grouped using boolean OR (on the assumption that they were in some sense equivalent) and the groups were then combined using the best-match operation. However this did not prove very effective, at least for the sample of users involved. (Those who enhanced their query with thesaurus terms representing two or more facets did so more successfully than those who dealt with only one facet. But the faceted *form* of their query tended to be less successful than a simple 'hybrid' of original query and additional thesaurus terms.)

Alternatively, we may treat faceting quite independently from probabilistic search. One suggestion is to introduce *filters*: conditions which come into operation after a query expression has generated a ranked document set. They do not change the composition or order of the hit-list, but modify the user's view of it by highlighting those entries which satisfy the extra conditions. So a document set corresponding to one facet is considered as a filter 'through' which another hit-list is viewed, the condition here being that the document occurs in both sets. This is roughly equivalent to boolean AND but has the advantage of retaining the original

probabilistic ranking, allowing users to see the consequence of applying different filters without re-conducting complete searches.

The introduction of a filter mechanism will have many implications both for the query model and for the interface, which we have not yet begun to explore. But it could perhaps support several of the non-probabilistic 'devices' mentioned earlier in this section, allowing users to fine-tune their view of the results without affecting the performance of the probabilistic algorithms which perform the really vital retrieval functions.

THE INTERFACE : OVERVIEW

Given the above query model, it is now comparatively easy to define a set of interface objects as simple views on the underlying objects. Our interface consists of a single window in which each such object occupies a region which may dynamically grow or shrink, appear or disappear, according to the current state of the search. We have:

- **Query term-list**: for words and phrases which form part of the current query. These terms appear in weight order, so that users can see which will be most influential in subsequent searches. The source of the term (user, thesaurus or relevant documents) will be indicated by colour-coding.
- **Candidate term-list**: for words and phrases found by the system in the thesaurus or in relevant documents, and offered as suggestions to the user.
- **Document hit-list**: for titles of documents found by the latest search, presented in weight order. Those seen and judged irrelevant by the user will be marked accordingly. Those which appeared in previous hit-lists during the same session will also be marked, indicating whether they have been 'promoted' or 'demoted' by the new search.
- **Displayed-document**: for the full text of a document being examined by the user (temporarily overlaying the current hit-list). Query terms in the text are highlighted, and a set of buttons is presented for the user to make a relevance judgement on the document.
- **Document pool**: for titles of documents judged relevant by the user.
- **Thesaurus**: appearing only when the user requests a thesaurus search, and temporarily overlaying both hit-list and document pool list. Within

this region are sub-regions to hold: lists of broader, narrower and related terms found in the latest expansion move, and a 'pool' of terms selected by the user for inclusion in the current query.

Selection and de-selection within all the above lists is performed by clicking or double-clicking on the relevant item. Handling of separate query facets and the structured presentation of thesaurus terms has not yet been tackled, but we expect to fit them fairly easily into the above framework.

Other interface objects have no direct counterpart in the underlying model, but are concerned directly with interaction and control, i.e.

- **Entry**: where the user types in his or her original query. The entered character string is parsed to identify its constituent terms, which are placed in the current term-list. It is also used in an initial thesaurus search to identify any partial matches with controlled-language terms, which are placed in the candidate term-list.
- **Control-panel**: presenting the main system functions for user selection, i.e.
  - ♦ Initiate a document search with the current query terms,
  - ♦ Initiate a thesaurus search with a selected thesaurus term as entry-point,
  - ♦ Expand the current query by extracting and re-weighting terms from the relevant document pool,
  - ♦ Enter additional user-defined words and phrases,
  - ♦ Move back and forward through previous query 'states',
  - ♦ End the session.
- **Help**: presenting a brief message explaining the function of the interface object currently 'in focus'.

Flow of control within the interface is specified initially using *state-event* tables governing what actions are allowed at each stage in the search process, and therefore when and how control objects are to be activated and presented. This logic is then implemented using TCL. Figure 3 shows the thesaurus search stage of an example session, illustrating how some of the interface objects described above appear to the user.

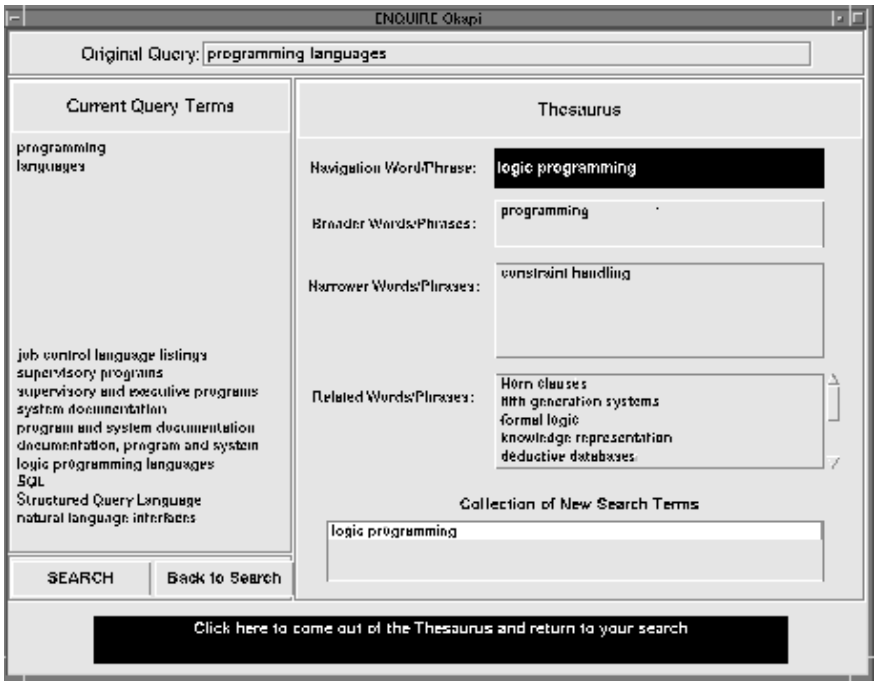

Fig 3: An example of the current interface

An important function for a system designed as an experimental tool is the logging of user actions and results. All meaningful events detected at the interface will be recorded on a log file with appropriate parameters, likewise summaries of document hitlists generated (perhaps by different search expressions based on the same term-list), and relevance judgements made. As in earlier Okapi projects[1,3], this data will be converted for storage and analysis by the Oracle RDBMS, allowing detailed investigations of the effectiveness of various search strategies.

ASSESSMENT OF DEVELOPMENT METHODS

An IR system is a complex piece of software which needs a clear structure as a basis for stable performance. One intended for experimental use also needs a high degree of flexibility, so that the value of individual strategies can be investigated without large-scale re-programming or the introduction of unpredictable effects elsewhere. And it is helpful if its users can be

given a correct but easily understandable view of what it contains and what it does, as an aid to effective searching.

Various development methods can be adopted to satisfy these aims. One involves the use of KBS techniques, where different parts of the system are personified as 'experts' in particular tasks, who communicate via a common work-area known as a 'blackboard'. Rules for controlling such systems are often coded in a high-level declarative language (e.g. Prolog) so that they are comprehensible and readily adaptable.

Our own chosen combination of techniques was also intended to help us achieve stability, adaptability, and ease of understanding. At the programming level the O-O approach demands very explicit definition of the way system components behave and co-operate (and generates correspondingly high volumes of code!). But the metaphorical framework of objects and properties makes it easy to communicate ideas at various levels of technical detail within a team of researchers, and we hope that this will also be apparent at the user interface. An interface which makes visible (though in a simple way) the underlying model should prove ultimately more helpful for users than a facade, however attractive, which obscures the structure behind it.

The TCL language also has repaid its quite considerable initial set-up and learning costs by the flexibility which it offers to designers and experimenters. With TCL it is possible to adjust any aspect of the interface, whether to change the colour of a piece of displayed text or to switch between user-directed and system-directed query expansion, even, if desired, while a retrieval session is running. As a typical Unix shell language, it supports the definition of new commands, and hence the construction of complete special-purpose command languages. While the main focus of our project is interface design for experimental purposes, a useful side-effect may be the development of a *high-level interface description language* for OKAPI, which can be used to generate new versions of the system quickly for different user groups and document databases.

Acknowledgements

- The ENQUIRE Project is funded by the British Library Research and Development Department.
- We wish to thank the anonymous referee who drew our attention to other applications of O-O design principles in Information Retrieval.

REFERENCES

1. ROBERTSON,S.E, WALKER,S, JONES,S, HANCOCK-BEAULIEU,M. *and* GATFORD,M. Okapi at TREC-3. D. Harman (ed). *The Third Text REtrieval Conference (TREC-3)*. Gaithersburg MD: NIST, 1995. (To be published).
2. HANCOCK-BEAULIEU,M. Enhanced Navigation and Querying at the User Interface for Retrieval Effectiveness in OKAPI (ENQUIRE OKAPI). *Research proposal to British Library Research and Development Department*, 1994.
3. JONES,S. GATFORD,M. HANCOCK-BEAULIEU,M. ROBERTSON,S.E. WALKER,S. *and* SECKER,J. Interactive thesaurus navigation: intelligence rules OK?? *Journal of the American Society of Information Science* 46, 1995, 52-59.
4. HANCOCK-BEAULIEU,M. Interactive query expansion in an OPAC: interface and retrieval issues. To appear in *Journal of Document and Text Management*.
5. HARPER,D. *and* WALKER,A.D.M. ECLAIR: An extensible class library for information retrieval.*The Computer Journal,* 35(3), 1992, 256-267
6. GOLDBERG,A *and* ROBSON,D. *Smalltalk-80 : The Language.* Addison-Wesley 1989.
7. OUSTERHOUT,J.K. *TCL and the Tk Toolkit*, Addison-Wesley, 1994
8. KEEN,E.M. Designing and testing an interactive ranked retrieval system for professional searchers. J*ournal of Information Science* 20(6) 1994, 389-398.
9. JONES,S. *and* HANCOCK-BEAULIEU,M. Support strategies for interactive thesaurus navigation. H. Albrechtsen and A. Oernager (eds). *Knowledge organization and quality management. Proceedings of a conference,* Copenhagen, 1994. Frankfurt: Indeks Verlag, 1994, 366-373.