Working Paper Series
ISSN 1170-487X

VQuery: a Graphical User Interface for Boolean Query Specification and Dynamic Result Preview

by Steve Jones

Working Paper 98/3
March 1998

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

VQuery: a Graphical User Interface for Boolean Query Specification and Dynamic Result Preview

Steve Jones
Department of Computer Science
University of Waikato
Private Bag 3105
Hamilton, New Zealand
Tel: +64 838 4490
E-mail: stevej@cs.waikato.ac.nz

ABSTRACT
Textual query languages based on Boolean logic are common amongst the search facilities of on-line information repositories. However, there is evidence to suggest that the syntactic and semantic demands of such languages lead to user errors and adversely affect the time that it takes users to form queries. Additionally, users are faced with user interfaces to these repositories which are unresponsive and uninformative, and consequently fail to support effective query refinement. We suggest that graphical query languages, particularly Venn-like diagrams, provide a natural medium for Boolean query specification which overcomes the problems of textual query languages. Also, dynamic result previews can be seamlessly integrated with graphical query specification to increase the effectiveness of query refinements. We describe VQuery, a query interface to the New Zealand Digital Library which exploits querying by Venn diagrams and integrated query result previews.

KEYWORDS: dynamic queries, query previews, query by diagram

INTRODUCTION
Digital libraries and other common on-line information repositories must provide effective access to their contents for a wide variety of users. In this paper we focus on user interface techniques to improve a particular mode of access–searching–and describe our work in developing an alternative user interface for the New Zealand Digital Library (NZDL) [18].

When searching, users specify terms of interest joined by query language operators, and information matching those terms is returned by an indexing and retrieval mechanism. World-Wide Web [3] based Internet search engines and some digital libraries (such as the NZDL) are examples of systems which provide textual languages for query specification. These languages commonly exploit Boolean logic, even though it has been shown that difficulties in dealing with Boolean logic are common, particularly when a restricted syntax is used [4, 9, 11]. The consequences are that significant numbers of erroneous queries are created. Beyond syntactic demands, the conflict between the meaning of operators in Boolean logic and English language poses problems. AND tends to be inclusive in English but is exclusive in Boolean logic; OR tends to be exclusive in English but is inclusive in Boolean logic. Alternative syntax is sometimes used. The union operator is commonly represented with |, +, or ∪; intersection by & or ∩; negation by ! or –. The lack of consistency in use of these operators across systems and their lack of direct relationship to their meaning creates further difficulties for users.

A further problem with systems such as the NZDL and most World-Wide Web (WWW) based search engines is their lack of responsiveness. Users are locked into a cycle of query refinement as shown in Figure 1, in which query specification and result browsing are distinct activities. The user receives no feedback concerning the effect of query refinements until a new set of results for the refined query is returned. The delay in returning results is dependent upon network performance, query complexity, loading of the information server and the volume of information to be searched. It may be substantial. Users can waste time and effort in forming queries which return zero or near-zero matching documents, or very large unmanageable numbers of matching documents. Additionally they receive no indication as to whether the query terms that they specify will identify documents of the appropriate nature.

We believe that a direct-manipulation user interface which exploits diagrammatic techniques for query specification can circumvent problems with textual query languages. Visual query notations can have two key advantages over textual query languages: they are less syntactically demanding and overcome the English language and Boolean operator ambiguity. We suggest that a particularly

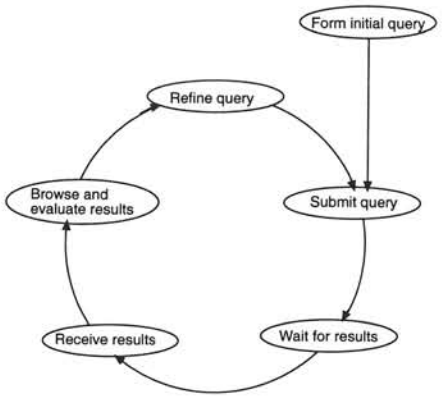

Figure 1: The query refinement cycle.

effective graphical notation is that of Venn diagrams which illustrate sets and their relationships. If such an interface also seamlessly integrated dynamically generated previews of query results users would be supported in applying more efficient searching strategies than are currently possible.

In the following section we describe related work. It provides evidence to support the use of Venn diagrams in query interfaces and discusses other work on dynamic query previews. We go on to describe the context in which our work is being undertaken – the New Zealand Digital Library. Next we describe in detail VQuery, a graphical user interface for query specification using Venn diagrams which provides dynamic query previews. To conclude, we discuss the utility of VQuery and outline our plans for future work.

RELATED WORK

Venn Diagrams as an Interface to Boolean Query Specification

The idea that Venn diagrams may provide an effective medium to help users specify Boolean expressions is not new. Halpin [10] convincingly argues that Venn diagrams are highly expressive, can represent a broad range of queries, 'give a clear picture of the meaning of the query' and 'provide a simple means of clarifying what set comparisons mean'. As long ago as 1976, Thomas [16] studied both interpretation and generation of Venn diagrams with a view to guiding the design of query systems.

Michard [14] describes GQL (Graphical Query Language), one of the first efforts to exploit Venn diagrams in database querying software. GQL was compared to TEST, a textual Boolean query language. In a comparative study subjects made almost four times as many errors when using TEST than when using GQL. Just under half of the errors with TEST were due to the use of incorrect syntax, and a fifth due to the use of an incorrect Boolean operator. Syntactic errors were not possible in GQL, and half the number of incorrect operators were used with GQL than were used with TEST. Complex Boolean expressions were more accurately represented in GQL than TEST.

Michard's results have been confirmed by other studies. Davies and Willie [6] carried out a comparative study of a Venn diagram user interface and a simple Query By Example [19] tool (QBE). They found that use of the Venn diagram query tool resulted in fewer errors, substantially faster specification of queries, and more positive user feedback. Hertzum and Frøkjær [11] compared Boolean retrieval using a textual query language and Venn diagrams. The Venn diagram interface was significantly faster and produced significantly fewer errors than the textual query language. Almost ten times the number of syntactically incorrect queries were produced using the textual language than with the Venn interface.

Katzeff [13] considered Venn diagrams in the context of users' mental models of information structures. Subjects in a study were given descriptions of a database and operations on its contents which adopted one of four models (no model, tables, shallow set explanation and deep set explanation). The set models were explained using Venn diagrams. The results show that the Venn diagram based models were more effective when subjects had to form complex queries requiring problem solving.

An important aspect of the use of Venn diagrams is reported by Willie and Bruza [17]. They identified *set assembly* and *set refinement* techniques as being equally common when subjects were able to formed Venn diagrams. In set assembly, intersections between sets are created by overlapping circles. In set refinement, circles are wholly contained within other circles to indicate intersections.

These studies indicate that a Venn diagram based user interface shows promise in more effectively supporting query specification than the standard textual query languages that are currently used. In particular, the syntactical constraints of a textual language are alleviated, and a reduction in erroneous queries and time taken to form queries is likely. The use of Venn diagrams also appears to alleviate, but not remove, the conflict between the meaning of Boolean AND and OR operators and the meaning of these terms in English.

Dynamic Queries and Query Previews

The concepts of dynamic queries and query previews both support users in effective query formulation. Dynamic queries are principally concerned with the provision of immediate feedback resulting from amendments to query parameters. Shneiderman defines a dynamic query to involve "the interactive control by a user of visual query parameters that generate a rapid (100ms update), animated, visual display of database search results" [15, p70]. In essence, users see the results of query refinements as they make them. This has several reported advantages: users gain a sense of control over the database, patterns in data can be quickly perceived, and new queries can be generated based on what is discovered through incidental learning

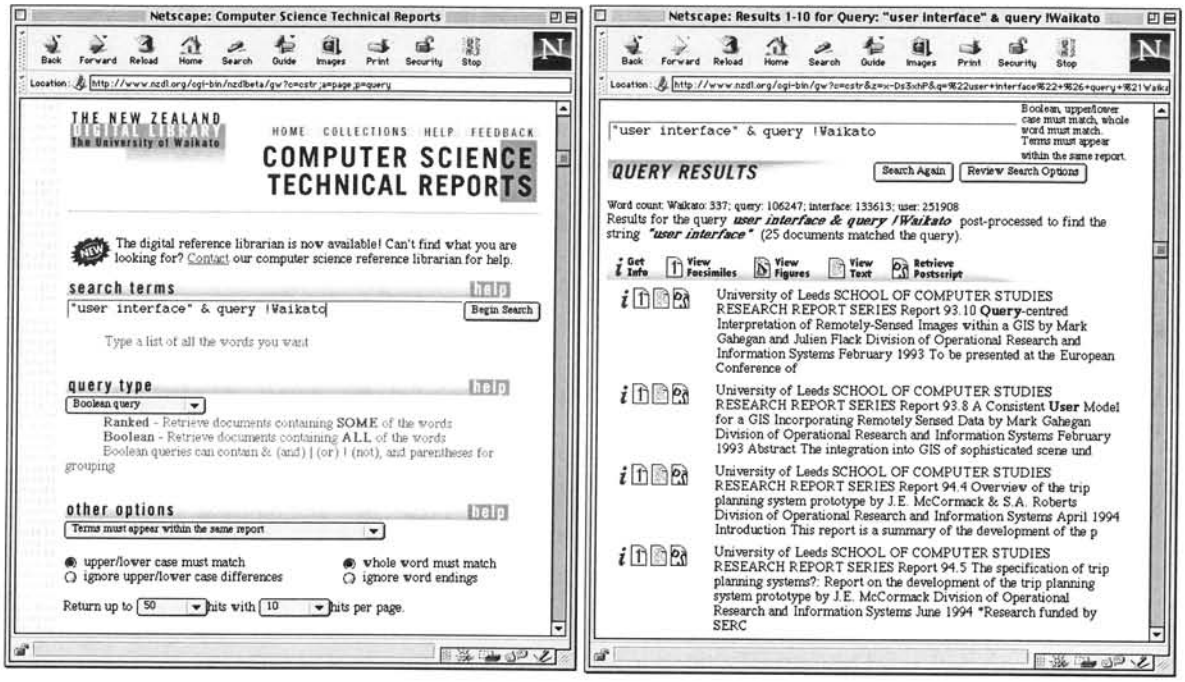

Figure 2: The current NZDL interface, showing a query screen (left) and a results screen (right).

[15]. Dynamic queries also result in tangible performance gains. Ahlberg et al [1] found that subjects formed queries significantly faster using dynamic queries than with a text based form-fill query interface. Conventionally, query parameters are controlled by slider user interface components, and one slider corresponds to one attribute of the database to be searched.

Problems with this approach have been highlighted by Fishkin and Stone [8]. In particular, only queries involving conjunction of terms are possible dynamically. Also the number of selectors, and consequently the number of possible queries, is fixed in advance. Fishkin and Stone overcame some restrictions of dynamic queries through movable filter windows, each expressing data values, which can be overlapped above a data space. However, this and previous approaches deal with structured databases with predetermined attributes, each attribute having a range of possible values that can be easily represented using sliders. We are concerned with querying of full-text indexes, and so must consider how the concepts of dynamic queries can be transferred to that domain.

Query previews are intended to overcome problems users experience with network performance, data volume and data complexity when forming and refining queries [7]. Their aim is to allow users to reduce the size of the set of returned items to a manageable size before the query is submitted across the network. This enables users to avoid empty result sets or overwhelmingly large result sets without the overhead of executing queries across the network. This is achieved by allowing users to specify rough ranges of values for rapid, iterative query refinement. Response from the system is immediate, and indicates the size of the result set matching the query.

In order to provide immediate response without the network overhead, this approach is dependent upon database providers publishing accessible tables of contents for searchable databases. The tables of contents must be small enough to be stored in high speed storage to enable dynamic querying. These are drawbacks of the query preview approach adopted by Doan et al [7]. Additionally, their preview interface displays only the number of matching items to the user. It provides no information to support the user in determining the relevance of returned items, which is a key issue in document retrieval.

Therefore, although dynamic queries and query previews hold promise for use in digital library user interfaces, we must consider how they may be implemented for a full-text retrieval system.

THE NEW ZEALAND DIGITAL LIBRARY

The New Zealand Digital Library (NZDL) is a freely-accessible system on the WWW[1] that provides full-text indexes to collections of documents. It is based in the Computer Science Department at the University of Waikato. Ten collections are freely available and several more are available only to specific end-user organisations. The freely available collections include an independently published weekly on-line newsletter, public domain literary works, information relating to indigenous peoples of the world and computer science technical reports (CSTR). The

[1] http://www.nzdl.org

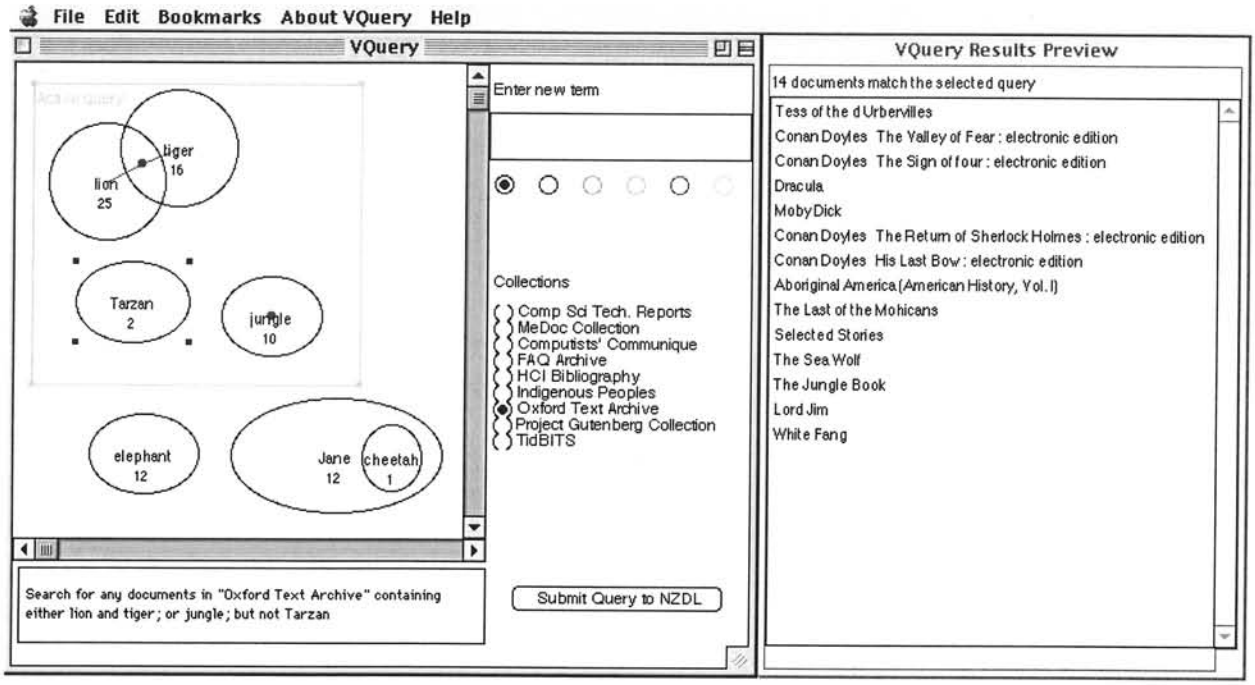

Figure 3: The VQuery user interface showing the query workspace (left) and the results preview window (right).

CSTR collection alone provides a full text index to approximately 40,000 technical reports gathered from over 300 sites world-wide, totalling 34 Gigabytes of source text.

The NZDL supports both ranked and Boolean querying, although it is the user interface to Boolean query specification that we address here. As with most other WWW based information sources users are required to form a combination of terms and operators in a particular syntax to represent their intended query. Intersection, union and complement operations are supported and are represented by &, | and ! respectively. Query terms or phrases are joined using these operators, and components of complex expressions may be grouped using parentheses. Queries may be further refined by the use of options. One option is granularity of the search. For example, multiple terms may be required to appear within the same report, same page or same paragraph. Other options control case-folding, stemming and the maximum number of matching documents to return. It is also possible to weight the terms within an expression, so that they have stronger or weaker effects on the documents that are returned. Sample query and result screens are shown in Figure 2.

VQUERY

We have developed a graphical query interface application called VQuery. The four main objectives of the system are to:

- provide an alternative interface to textual Boolean query specification;
- dynamically provide responses to refinements of queries;
- support more effective searching through provision of query result previews;
- support reuse of queries through both their short-term and long-term storage and retrieval.

VQuery is an alternative to the textual query interface currently used by the NZDL. Although used in conjunction with the NZDL at the moment, it is easily portable to other digital library or search engine applications which operate over the WWW.

The VQuery User Interface

Figure 3 shows the VQuery interface. It has two main components: the query window and the result preview window. The query window is divided into three main areas. To the top left is the *query workspace*. This is a window in which the user can organise query terms to create Boolean queries. To the bottom left is the *natural language query view* – a text panel in which an English interpretation of the current query is displayed. To the right is the *control panel*.

A Workspace for Query Manipulation. The query workspace is a large scrollable window, which when fully expanded provides a full screen workspace. It displays circles containing query terms which may be phrases or single words. Each circle may be thought of as the set of documents containing the term within the circle, and below each term is a number indicating how many documents are in that set. A new term is created by typing a word or phrase into the term entry box within the control panel and the number of terms that can be added to the workspace is unlimited. The circles can be selected (as 'Tarzan'

currently is), moved and resized using standard pointing and dragging with a device such as a mouse. Multiple terms can be selected, moved and resized using a keyboard modifier. Selected terms can be removed using the delete key. Set assembly and refinement representations are both supported – terms can overlap (such as 'lion' and 'tiger' in Figure 3) and be fully contained within other terms (as 'cheetah' is within 'Jane' in Figure 3)..

The workspace is divided into two areas; an active query area within the rectangle labelled 'Active query' and a non-active query area outside of it. The active rectangle contains the terms that the user is combining to form a query and the non-active area acts as a storage area for terms which have been used in the past or may be used later, but do not contribute to the current query. Terms can be dragged in and out of the active area to amend the active query. The active query area can be moved or resized by dragging its bounding rectangle, providing a second method for inclusion or exclusion of terms.

Through the arrangement of term circles and positioning of the active query area users can create complex queries containing multiple terms and using combinations of the AND, OR and NOT operators. An AND operation on two or more terms is represented by query-selection[2] of the intersection of the circles of those terms. An OR operation on two or more terms is represented by query-selection of the circles of those terms but not in the area of their intersection. Any term circles within the active area which are not query-selected are combined with other terms using the NOT operator.

Query-selection of terms can be either transient or fixed. Transient selection is used when immediate previews of results of queries involving single terms or the AND operator are required. In this case, when the user moves the mouse pointer into a circle a dot (which is red in the interface) will appear at its centre indicating that it is to be used within the query being formed ('jungle' in Figure 3 is transiently selected). The results preview window will be updated to contain the titles of the documents which contain the term within the circle. When the mouse pointer leaves the circle, the dot is removed and the document titles associated with the term are removed from the results preview window.

If the mouse pointer is moved into the intersection of two or more circles a dot appears at the centre of their intersection, and the circles are joined by lines, to represent their association in the AND operation. Again the results preview window is updated, and the interface reverts to its prior state when the mouse pointer leaves the intersection.

When the user is satisfied with a component of the query its query-selection can be fixed by clicking within its circle or the intersection of multiple circles. In this case the query-selection is not removed when the mouse pointer leaves the selected region. This is useful for experimentation with addition of other components to a base query, and is necessary to allow specification of the OR operator using disjoint term circles. Fixed query-selections can be removed by a second click within the region of the selection.

Result Previews. The result preview window contains a label which indicates how many documents match the currently selected query and a list of titles of the documents which will be returned by the currently selected query. The current query may be refined through transient selection, fixed selection, term deletion or manipulation of the active query area, and in each of these instances the result preview is immediately updated to reflect the change. Single or multiple selections of items within the title list can be made using the mouse and a keyboard modifier. When the user double-clicks on selected items, the full text of those items is retrieved from the NZDL and displayed to the user.

Query Reinforcement. The natural language query view is situated below the query workspace. It presents an English language interpretation of the active query, which for new users reinforces the semantics of the visual notation, and provides feedback as to the meaning of a selected query. It is immediately updated whenever the active query is amended. We use a simplistic transformation, attempting to reduce ambiguity in the English expression. English versions of Boolean expressions are inserted into boilerplate text which gives user instructions and indicates which collection is to be queried. Comma separated lists of terms are used for AND and OR operations on multiple terms. Compound queries containing more than one multiple term operation are represented as semi-colon separated lists.

Storage and Retrieval of Queries. Users are provided with utilities to store and retrieve VQuery state both within and between query sessions. To save the current state of a query workspace the "Bookmark current state..." command is used, accessible through the Bookmarks menu. When this is selected the user is prompted to supply a textual label for the bookmark. This label is then added to the Bookmarks menu. There is no limit on the number of bookmarks which can be created in a VQuery session. When a bookmark is selected from the menu the current workspace state is replaced by the state at the time that the bookmark was created. The replication of the state is exact, including size, positioning and selection of all components of the query workspace. Each bookmark is stored in an ASCII file exemplified in Figure 4.

Persistent storage of a querying session is also supported. Commands to save and load sessions are accessible through the File menu and users are prompted with standard file selection dialogs to name and select sessions. The file format for session storage is very simple, merely containing the names of the bookmark files associated with the session

[2] We make a distinction between selection of terms for inclusion within a query ('query-selection', represented by dots) and standard selection for editing (represented by small black rectangles).

```
collection      <Oxford Text Archive>          #active collection at time of save
active-query    <12><14> <224><206>            #location and dimensions of active query area
term-count      <7>                            #total number of terms
term            <Tarzan> <42><136> <78> <55> <black>        #term, location and size, and colour
term            <jungle> <142><146> <68> <54> <black>
term            <elephant> <52><240> <75> <54> <black>
term            <Jane> <149><229> <143> <78> <black>
term            <cheetah> <238><247> <40> <45> <black>
term            <tiger> <72><18> <80> <80> <black>
term            <lion> <23><41> <80> <80> <black>
model-active    [<Tarzan><jungle><tiger><lion>]      #terms contained within active query area
model-result    [[<lion><tiger>][<jungle>]]          #all terms currently containing red dot
fixed-result    [[<lion><tiger>][<jungle>]]          #all fixed selection terms
                                                     #intersecting terms grouped together
end
```

Figure 4: ASCII file format for saved bookmarks.

and a record of which bookmark was active when the session was saved.

Utilities. The control panel provides three functions. At the top right users can enter new query terms into a text entry area. When the RETURN key is pressed a circle containing the term appears in the query workspace. Below the text entry area are radio buttons which allow the user to select the default display colour for newly created terms. Both the circle and the text of new terms are displayed in the selected colour. Below the colour selection panel, users can select which information source will be queried. A variety of document collections from the NZDL are currently displayed. When a new information source is selected the natural language view is immediately updated to reflect the change. Below the information sources is a single button "Submit Query to NZDL" which submits the active query to the NZDL via the user's WWW browser. This means that the user can get the standard HTML result format from the NZDL should they require it.

The VQuery menu bar contains three action menus (File, Edit, Bookmarks) and two menus for accessing information about VQuery (About VQuery and Help). Standard editing commands (Undo, Cut, Copy and Paste) for use in manipulating term circles within the query workspace are provided and these are accessed through the Edit menu. It contains two additional utilities which saves the user from carrying out common actions term by term. The first is "Clear Selections" which removes all currently fixed selections within the query workspace, and second is "Clear Workspace" which removes all term circles from the query workspace.

VQuery implementation and architecture

VQuery is implemented as a Java application which may be used in place of, or in conjunction with the conventional NZDL user interface. Using the conventional WWW interface to the NZDL, the user creates a textual query via HTML forms and submits it. The query terms and options are used to form a URL which invokes a CGI script on the NZDL server with the appropriate parameters. The CGI script communicates with full-text indexing and retrieval software and a set of documents matching the query is formed. The document details are inserted into boilerplate HTML which is returned to the user's browser to display the results of the submitted query. Although our indexing mechanism MG [2] is efficient for very high volumes of information, response to the user is never immediate due to variables such as network performance, data volume [7], server loading and query complexity. This process can be seen in the lower portion of Figure 4.

To provide dynamic result previews VQuery communicates directly with the NZDL server, and to allow display of results in standard NZDL format VQuery communicates with the user's WWW browser. This can be seen in the upper portion of Figure 4. When the user creates a new term, a background process is started to retrieve matching documents. This process connects to the NZDL server sending a URL representing a query containing the single term. As with standard use via a WWW browser, matching documents are returned in HTML format. The background process parses the HTML, extracting the details of the document titles and the number of matching documents. Whilst this is taking place a circle containing the term is placed in the query workspace, and the user may continue to manipulate terms. Once the results have been obtained, the number of matching documents is displayed within the term circle, and the result preview window is updated. Although the delay in result retrieval is the same as with conventional WWW access it is much less evident to the user because VQuery remains interactive. Usually, by the time that initial placing of the new term circle has been completed the results have been returned. A result set is associated with each term and when transient or fixed queries are created the corresponding operations are applied to the result sets of the terms involved, providing a list of matching document titles for display in the result preview window. In fact, a result set for each collection can be associated with each term, so that the user can investigate the result of queries in multiple collections. Currently, to achieve this it is necessary to produce a parser for the results from each different collection, because the result format differs between collections. However, a new

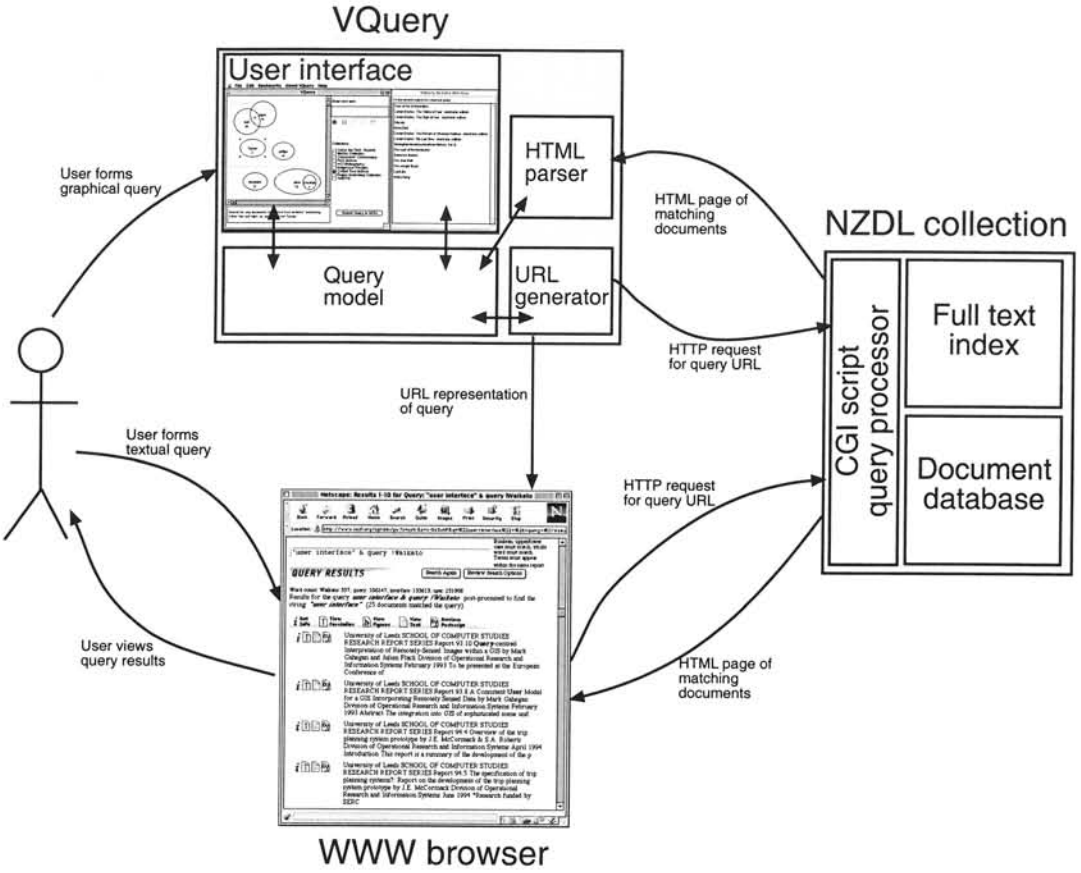

Figure 4: The interface between VQuery, the user, a WWW browser and the NZDL.

architecture for the NZDL will soon allow results to be retrieved in a standard format, and only a single parser will be required. When bookmarks are recalled within a session, there is no need to retrieve results from the NZDL because they are retained in VQuery for the duration of the session. However, when sessions are recalled results are retrieved from the NZDL for each term in the session. This is because the NZDL collections may have been updated since the user's previous session, and the most current holdings should be provided to the user.

When the user submits a query to the NZDL from within VQuery, to take advantage of the NZDL display formatting, the user's WWW browser is directed to load a URL matching the query. Currently the user might use either Netscape Navigator or Microsoft Internet Explorer, but the way in which they can be controlled from external processes is both platform and version dependent. Ideally, VQuery would contain an HTML browser window.

The VQuery software architecture is flexible, adopting a Model–View–Controller approach [5]. An underlying data structure *models* the list of terms created by the user, and the semantics of the application. This is necessarily abstract so that it is sufficiently independent of particular user interface designs. The *View* component presents the current state of the model to the user on the display. The *Controller* component deals with user input. The View and Controller components are closely linked in a direct manipulation interface such as VQuery. In fact, VQuery uses four View–Controller pairs which integrate with the single data model. The first is the query workspace which displays created query terms and indicates term selection. This also supports user input. Circles are moved, resized, selected and so on. The second pairing is the natural language query output. Essentially this is only a view, with an inactive controller component. Third is the URL corresponding to the active query which will be sent to the NZDL server. Again this has an inactive controller component, and is, in fact, hidden from the user. The final view of the model is the results preview window.

This architecture supports provision and investigation of extended or alternative interfaces. Other View–Controller pairings may be integrated with the model.

DISCUSSION

VQuery implements a visual representation for query specification based on Venn diagrams, which have been shown in previous studies to be an effective alternative to

more common textual query interfaces. Our representation diverges from strict Venn diagram notation, because in previous studies we have found that users are comfortable with a Venn-like representation and tend not to utilise a strict representation [12]. The representation supports both set assembly and set refinement techniques which our previous studies and the work of others have shown to be commonly used.

There are limitations of representation which we must address. For instance, there are some Boolean expressions that can not be represented using VQuery, mainly because it does not support selection of the universal set. In this context we consider the universal set to be all document within a collection. Consequently some complement expressions such as those involving the complement of single sets, or the complement of compound expressions are unavailable to users. This means that users are unable to request 'documents that do not contain A', where A is a single term or more complex expression. Although less than 2% of Boolean queries in the NZDL use the complement operator it is important that its provision in VQuery is complete and consistent. We must consider how best to support this in the VQuery interface.

The visual language for queries becomes unwieldy when intersection of more than three terms is required. It is difficult to organise four overlapping circles or ellipses to enable selection of all possible intersections. 19% of queries in the NZDL use more than three terms and so it is important to support user specification of intersection of more than three terms. One approach may be to provide some abstraction technique which allows multiple query terms to be collapsed into a single graphical entity. Michard [14] suggested this but found it to be responsible for the introduction of user errors, and so its design must be carefully considered.

The dynamic result previews support more efficient search refinement than conventional interfaces. These conventional interfaces give no indication as to whether a query might return empty or very large result sets until the results themselves are provided, which imposes a delay. VQuery supports the user in deciding whether both the volume and nature of the documents that *would be returned* are appropriate without that delay. The volume is indicated by the inclusion of the number of documents matching single terms within their term circles. Users can immediately determine whether or not they might use a broader or more specific term, or consider using a synonym. When terms are combined to form a query, the user can determine if the query will provide a useful and manageable number of documents from the number of matching documents indicated in the results preview window. The titles of matching documents can be browsed to determine if the nature of the results matches user requirements. Through these facilities users receive immediate feedback regarding the efficacy of the queries that they have formed.

The flexibility of the query workspace has not been provided in previous diagrammatic query techniques. It can be used to hold both active query terms and those which the user wishes to put aside for the moment. In the conventional NZDL interface, and the interfaces of similar services, only those terms to be used in the query are visible at any one time. As a process of query refinement takes place it becomes more difficult for users to recall the set of query terms used and the ways in which they have be combined. In VQuery, the query workspace can serve as an external representation of the user's short term memory. The flexible way in which term circles and the active query area can be moved and resized provide for easy inclusion and exclusion of terms and encourages experimentation at little cost to the user. The range of terms used during a query session does not have to be recalled, merely recognised, reducing the cognitive demands when a complex sequence of queries and refinements is being formed.

A simple supportive facility is the allocation of colour to individual query terms. This allows users to create visual prompts regarding the use of terms within the workspace. For example, a user might allocate a particular colour to all terms related to a single concept. Alternatively, a user might relate colour to the primary collection in which the term is to be searched for.

The facility to save the current state of a query session using bookmarks encourages user experimentation with queries, knowing that prior work is recoverable. Backtracking is simplified and the states resulting from backtracking are more evident than those reached by backtracking through HTML form based interfaces. Bookmarking can also be used to support investigation of multiple threads of queries within the same session, and allow attention to be easily transferred from one thread to another. The facility to save collections of queries and bookmarks to persistent storage recognises that users might not be able to carry out an exhaustive search to meet their information seeking goals within a single session. Reloading of sessions also reduces the repeated effort that is commonly required by standard query interfaces in order to return to previous query session states. User may wish to do this to maintain awareness in a particular topic or identify new holdings.

FUTURE WORK

Our immediate work will concern the identification of an appropriate method for universal set selection so that the complete range of Boolean queries may be expressed in VQuery. We will also provide an abstraction mechanism to allow relationships between multiple terms to be represented by a single entity in the query workspace. It will be necessary to determine if the errors that Michard reported to be introduced by such an approach can be avoided.

Of primary concern will be a comparative study of the conventional query language of the NZDL and the graphical approach adopted in VQuery. We are interested

in the comparative accuracy and speed of the two interfaces in allowing user to form queries which match their information seeking needs. Suitable sample queries will be identified to allow comparison with previous studies and allow consideration of the utility of our diagrammatic representation, flexible diagram layout, storage and retrieval of queries, natural language feedback and dynamic result previews.

ACKNOWLEDGMENTS

Thank you to Shona McInnes who developed an elegant and robust initial Java implementation of VQuery, based on prototypes implemented by the author; and who also carried out an enlightening user survey.

REFERENCES

1. Ahlberg, C., Williamson, C., and Shneiderman, B. Dynamic Queries for Information Exploration: an Implementation and Evaluation. In *Proceedings of CHI'92: ACM Conference on Human Factors in Computing Systems* (Monterey, California, USA, 3-7 May, 1992), ACM Press, pp 619-26.

2. Bell, T.C., Moffat, A., Witten, I.H., and Zobel, J. The MG Retrieval System: Compressing for Space and Speed. *Communications of the ACM*, 38,4 (April 1995), 41-42.

3. Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H.F., and Secret, A. The World-Wide Web. *Communications of the ACM*, 37, 8 (August 1994), 76-82.

4. Borgman, C.L. The User's Mental Model of an Information Retrieval System: an Experiment on a Prototype Online Catalog. I*nternational Journal of Man-Machine Studies*, 24, 1 (1986), 47-64, 1986.

5. Burbeck, S. Applications Programming in Smalltalk-80: *How to Use Model-View-Controller (MVC)*. Softsmarts Inc., 1987.

6. Davies, T. and Willie, S. The Efficacy of a Venn-based Query Interface: an Evaluation. In *Proceedings of QCHI95 Symposium*, (Bond University, Queensland, Australia, August ,1995), pp 41-50.

7. Doan, K., Plaisant, C., Shneiderman, B., and Bruns, T. Query Previews for Networked Information Systems: a Case Study with NASA Environmental Data. *SIGMOD Record*, 26, 1 (1997), 75-81.

8. Fishkin, K. and Stone, M.C. Enhanced Dynamic Queries via Movable Filters. In *Proceedings of CHI'95: ACM Conference on Human Factors in Computing Systems* (Denver, Colorado, USA, 7-11 May, 1995), ACM Press, pp 415-20.

9. Greene, S.L., Devlin, S.J., Cannata, P.E. and Gomez, L.M. No IFs, ANDs or ORs: a Study of Database Querying. *International Journal of Man-Machine Studies*, 32, 3 (1990), 303-326.

10. Halpin, T.A: Venn Diagrams and SQL Queries. *The Australian Computer Journal*, 21, 1 (1989), 27-32..

11. Hertzum, M. and Frøkjær, E.: Browsing and Querying in Online Documentation: a Study of User Interfaces and the Interaction Process. *ACM Transactions on Computer-Human Interaction*, 3, 2 (1996), 136-161.

12. Jones, S. and McInnes, S. A Graphical User Interface for Boolean Query Specification. *Working Paper 97/31*, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 1997. *Submitted to International Journal on Digital Libraries.*

13. Katzeff, C. The Effect of Different Conceptual Models Upon Reasoning in a Database Query Writing Task. *International Journal of Man-Machine Studies*, 29, 1 (1988), 37-62.

14. Michard, A. Graphical Presentation of Boolean Expressions in a Database Query Language: Design Notes and an Ergonomic Evaluation. *Behaviour and Information Technology*, 1, 3 (1982), 279-288.

15. Shneiderman, B. Dynamic Queries for Visual Information Seeking. *IEEE Software*, 11, 6 (1994), 70-7.

16. Thomas, J.C. Quantifiers and Question Asking. *IBM Research Report 7021976*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1976.

17. Willie, S. and Bruza, P. Users' Models of the Information Space. In *Proceedings of SIGIR95, the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, (Seattle, Washington, July 1995), pp 205-210.

18. Witten, I.H., Cunningham S.J. and Apperley M.D. The New Zealand Digital Library Project. *New Zealand Libraries* 48, 8 (1996), 146-152.

19. Zloof, M.M.: Query By Example. *IBM Systems Journal*, 16, 4 (1977), 324-43.