TELIDON

TELIDON BEHAVIOURAL RESEARCH 5

Interactive query languages for external databases

Interactive Query Languages for External Data Bases

F.H. Lochovsky

D.C. Tsichritzis

March, 1981

Computer Systems Research Group
University of Toronto
Toronto, M5S 1A1
Canada

For: Department of Communications, Ottawa, Canada
Under: Department of Supply & Services Contract Serial No. 0SU80-00124
Period: July 1, 1980 - March 31, 1981

DD 3640915
DL 3642465

Copies of

TELIDON BEHAVIOURAL RESEARCH 5
INTERACTIVE QUERY LANGUAGES FOR EXTERNAL DATA BASES

are available
free of charge from:

Information Services
Department of Communications
300 Slater Street
Ottawa, Ontario
K1A OC8

Report Number DOC-TBR-DBRE-81-5-E
(Printed in March, 1982)

TK
7882
I6
T46e
#5

ABSTRACT

The amount of information that is available in computerized form is growing steadily. Much of this information is of an informational or reference nature and could potentially be made available to the general public. Until recently, there was no convenient medium by which the public could gain access to this information. However, two developments — the growth of communication networks and the advent of videotex systems — provide the potential for allowing the public convenient access to this information. In this report we explore the nature of the mechanism by which people in their homes using a videotex information service might access data bases external to the videotex system itself.

We first consider the process of accessing a data base (querying) in terms of the user interaction characteristics of the process. We divide the querying process into three parts: request, reply and dynamics. For all three parts, certain characteristics are identified that describe the parameters of the user interaction. For requests, the parameters are number of keystrokes required, type and number of commands available, degree of freedom in request formulation, selectivity of requests, uniformity of requests with respect to data bases accessed and ability to customize the request formulation for specific applications. For replies, the parameters are the form in which the reply is presented to the user, whether the reply can be presented via more than one medium, whether the reply can be customized, the degree of dynamic control over the reply and the possibility of saving the reply and using it as input at some later time. For the dynamics of interaction, the parameters relate to the bandwidth between the user and the system, the degree of gamesmanship involved in the interaction, number of protocols available for interaction, the responsiveness of the system to a user request and the degree of control experienced by the user in an interaction. For each of these parameters, a desirable "value" for the parameter is identified.

We then identify seven different query language types: keyword, by example, natural language, menu, graphic, multi-media and computer games. Each query language type is described and evaluated in terms of the interaction parameters.

From this evaluation of query languages, it is obvious that no one type of query language has all of the desirable characteristics for a query language. We therefore propose some requirements that a query language for external data bases should have. With these requirements in mind we then propose a design for a query language for accessing external data bases.

For describing the query language, a paradigm of navigating a personal spaceship through space using detailed maps for guidance is used. Basically the user is able to request maps of the information space available to him and to choose his destination from these maps. The maps are presented in the form of templates which show the structure and/or contents of the external data bases. Most of the interaction is accomplished by pointing at fields of the templates and either filling in values or requesting a display of values. The nature of the interaction allows formatted data, nonformatted text data, image data and voice data to be manipulated in a uniform manner.

CONTENTS

1 INTRODUCTION

Recent years have seen a substantial growth in the number of on-line, computer-based information sources (data bases). Many of these on-line data bases provide information to the public, e.g., library systems, airline systems and government information services. However, the public generally does not have convenient and direct access to these data bases. Instead, some intermediary queries the data base and supplies the response, or the information is available in a non-selective manner in fixed locations.

Another development in recent years has been the installation and growth of communication networks. These communication networks allow data bases to be queried remotely at reasonable cost. This in turn raises the possibility of permitting access, by the public, to on-line "consumer information" data bases. An important question in this context is from where and how is this access to be permitted.

A third development in recent years has been the emergence of videotex systems. *Videotex systems* are interactive, visual communication systems intended, in part, to permit public access to external data bases. (By an *external* data base we mean one whose contents and format is not under the control of the information delivery (videotex) system.) Telidon is an example of such a system [Bown *et al.*, 1978]. The interaction with the system is via a suitably modified television receiver which acts as the terminal display. Input is accomplished by a keypad or keyboard which allows certain data to be selected for display.

Given that the public will eventually have access to videotex systems, several issues relating to user interaction with these systems arise [Bown *et al.*, 1979]. One of these issues concerns how the public will request information from the various external data bases that will be available to them. Currently, a number of different languages each with its own syntax and vocabulary exist for querying data bases.

It seems intuitively undesirable for a user of a videotex system to have to learn and remember a different query language for each external data base that will be accessed. To help ensure user acceptance of videotex information retrieval services, there should be one way of querying external data bases via videotex systems that is easy to learn and use by the public. Using this query language, people can access information in any external data base that is connected to the communication network.

In this report we identify and describe several categories of interactive query languages. We establish some user interaction criteria for evaluating these categories of query languages. We briefly discuss each category of query language with respect to these criteria. Finally, we propose a particular approach to a query language for external data bases.

2 USER INTERACTION PARAMETERS

The general subject of person-computer interaction has been the subject of much study [Martin, 1973; Gilb and Weinberg, 1977; Guedj *et al.*, 1980; Schneiderman, 1980; Mehlmann, 1981]. In the area of interactive query languages the studies are far fewer [Reisner, 1981]. Applying some of the results from general person-computer interaction and the studies on query languages, let us examine the process of querying a data base from the user's viewpoint. The querying process can be broken up into three parts: request, reply and dynamics.

Request deals with the formulation of a query to the system by the user. The user informs the system of some action that he wishes it to do for him. We are principally concerned here with requests for stored information. Usually some form of "language" that is mutually acceptable to the user and the system is used to specify a request. This language can take several forms, e.g., visual, verbal, pantomime, etc.

Reply deals with informing the user of the result of an action. The result can be the answer to a query or information about the status of a request. Presenting output to the user should be done in a way that is palatable to the user and easy for him to assimilate. Again, some form of "language" is required to communicate the output. Displayed output, either as tables, graphs, pictures, etc., seems to be the natural choice for videotex systems. However, the use of sound is also a possibility.

Finally, *dynamics* deals with the nature of the interaction between the user and the system. The request and reply processes must be accomplished via some communication medium between the user and the system, i.e., the "languages" must have some means of transport between the two "participants". For example, the interaction can be by typing, pointing, speaking, etc. The quality of this interaction, from the user's viewpoint, is a critically important aspect of the querying process. In videotex systems, we would like as much as possible that this interaction be easy and interesting for the user.

In most interactive query languages, these three issues are lumped together. This is a mistake since a language used for formulating requests has different user requirements than a language that is used for presenting replies. In addition, the dynamics of a language are independent of the static aspects of formulating requests or presenting replies. Thus requirements for the dynamics of a query language can be considered independent of request and reply requirements.

For each part of the querying process certain characteristics can be identified. These characteristics describe the parameters of user interaction. The "values" that these parameters have in a given query language determine whether or not the query language has "desirable" characteristics with regard to person-computer interface. In the rest of this section we will identify and define several characteristics for each part of the querying process. In addition, we will indicate what, in our opinion and that of the studies cited earlier, constitutes a "desirable" value for each characteristic. This list of "desirable" values will then constitute requirements for a "desirable" interactive query language.

2.1 Request

We identify six characteristics of formulating requests which are important for evaluating interactive query languages.

KEYSTROKES

Keystrokes are a quantification of the amount of user-supplied input required before the system fully understands the user's request. If requests are formulated by typing, then the number of characters that must be typed can be the measure used. If requests are formulated by pointing, then the number of things that must be pointed at can be used as a measure. It is desirable to *minimize* the number of keystrokes required to formulate a request. In this way requests can be formulated very quickly and easily and the possibility of some errors, e.g., typing errors, are reduced.

COMMANDS

Commands are the set of instructions a user has available to tell the system what action it should perform. For example, a user might want to tell the system to *retrieve* or *print* or *draw.* The number of commands available to a user should be *small* so that they can be remembered easily. The commands should also be *simple.* By this we mean that the syntax of a command and the number of things that need to be specified to use a command should be easy to remember and intuitive to the user.

FORMULATION

Formulation of a request concerns how difficult it is to specify a request that is incorrect either syntactically or semantically. A syntactically incorrect request is one that does not follow the "pattern" expected by the system. These usually consist of misspelled words or transposed or missing parts of a request. A semantically incorrect request is either one that cannot be answered by the system because it has no information about the request or one, for which the answer supplied, is not the reply the user expected. That is, the system's interpretation of the request differs from the user's interpretation of the request. It is desirable that a query language eliminate the possibility or make difficult the formulation of syntactically and/or semantically incorrect requests.

SELECTIVITY

Selectivity is the ability of the user to specify as precisely as possible that data which he wishes to retrieve. High selectivity of a query language implies that the user gets back only the data he wants. Low selectivity implies that much of the data that is retrieved is "noise" in that it is not relevant to the request. It is desirable that a query language have *high* selectivity. In this way the user is not distracted by irrelevant output. In addition, if the user is being charged according to the amount of data retrieved, then high selectivity may also minimize the cost of using the system.

UNIFORMITY

Uniformity concerns the independence of the query language from the type of data base it is accessing and the application for which the language is being used. Uniformity for data bases means, given several different types of data bases, some of which may be structured differently, can the query language be used to access all of these data bases. Uniformity for applications means, can the query language be used for any type of application or is the structure of a query application specific. For example, if a query language can be used to access stock, weather, news, grocery, etc. information, then it is uniform with respect to application. It is desirable that the structure of a query language be uniform with respect to both data bases and applications. In practice the latter is usually achieved easily; the former is much more difficult.

CUSTOMIZING

Customizing of a query language deals with matching the form of a query language to that of a particular application. The intent is to make it easier to form requests for the application since the form of the query language helps the user remember the language. For example, certain applications use specific words for specific actions. These words could be used as keywords in a query language. In this way, the semantics of a request are conveyed more easily and precisely to the user. It is desirable that a query language employ as many application specific concepts as possible to increase user convenience in interacting with the system..

2.2 Reply

We identify five characteristics for representing replies which are important for evaluating interactive query languages.

PRESENTATION

Presentation is the form in which the reply is given to the user. The important consideration here is the complexity of the presentation. One measure of this complexity is how long it takes the user to grasp the content of the reply. If the reply is presented in dialogue form which the user must read on a display, then it can take quite some time to understand the information content of the reply. On the other hand, tabular or graphical presentations can usually be interpreted much faster. In addition, much more information can be presented in the same amount of space. It is desirable that the query language display the reply to a query in a form which is acceptable to the user and which is simple for the user to follow.

MULTI-MEDIA PRESENTATION

People communicate in more than one way. Therefore, as well as the form in which a reply is presented to the user, it is also desirable that the user be able to choose the medium by which he receives the reply. For example, the reply could be printed on paper, be displayed either as text, graphics or animation on a screen, or be in spoken form. In this way, the way in which the reply is presented could be matched to the moods and requirements of the user. He may want to leisurely browse through the reply, glance at it quickly, or listen to it while doing other things. The ability to direct the output to different devices and to different communication media may also be important for allowing handicapped people to use the system.

CUSTOMIZING

Query languages usually do not take cognizance of the audience for which the reply to a request is intended. There is usually a fixed way of presenting the reply to the user irregardless of application, type of display device, time of day or type of user. Customizing of replies, as for customizing of requests, means that the form of the reply is determined by the environmental factors of the interaction. For example, weather information may best be displayed graphically rather than textually.

Children may want to get replies in terms of cartoons which they can understand rather then text or graphics which they find difficult to interpret. Users may want to get replies in different forms depending on the time of day, e.g., they may want to get short, verbal summaries late at night but more lengthy, printed information in the early evening. It is desirable that a query language provide the facilities to customize the replies given to a user dependent on factors such as those cited above.

DYNAMIC CONTROL

A user should be able to control the pace at which the reply to a request is presented to him. That is, given that the reply will be presented in a certain form via a certain medium, the system should provide the user with the means for dynamically controlling the presentation. The user should be able to control the speed at which text is presented on a display. He should be able to control this speed dynamically skipping over parts or going back in the text. He should be able to change the form of the reply, say from text to voice, dynamically without reformulating the request. In addition, these changes should be done fairly easily.

REUSABILITY

Often the reply to a request is not only for immediate consumption, but may be useful at a later time or as input in the formulation of another request. It is therefore desirable that the query language allow the reply to a request to be saved for further processing or to be redirected as input to another request. In this way, it is possible to break complex queries into simpler "intellectually manageable" units and to reduce the total amount of interaction between the user and the system. This manner of interaction may more closely approximate the way in which people handle complex tasks.

2.3 Dynamics

We identify five characteristics of the interaction between the user and the system which are important for evaluating interactive query languages.

BANDWIDTH

The bandwidth of interaction is a measure of the speed at which the user and the system communicate with each other. For example, it is usually the case that people can absorb more information if it is presented in pictorial form than if it is presented as written text. Typing a request to the system has a much lower bandwidth than being able to communicate the request verbally. Bandwidth is a function of the technology available for interaction as well as the way in which requests and replies are specified. It is desirable to *maximize* the bandwidth of communication between the user and the system via the query language.

GAMESMANSHIP

It is important that the interaction between the user and the system be interesting to the user. By this we mean that the nature of the interaction is challenging to the user and that the degree of difficulty of interaction is related to the benefit of the outcome. We call this parameter gamesmanship. Users of videotex systems have no external motivation, e.g., monetary, for interacting with the system. The motivation is purely internal, e.g., they want to learn or be entertained. The user must perceive that the investment of his time is worth the result obtained or at least that the time spent was enjoyable and not frustrating. Thus if formulating requests is interesting and challenging, then the tediousness of the interaction may be alleviated by its learning or entertainment value.

PROTOCOL

A protocol is a specified way in which a user interacts with the system. For example, a user may always formulate a request in the same way by typing it on a keyboard. It is desirable that a query language allow multiple protocols for user/system interaction. In this way, the user can choose the protocol desired at any particular time. It may be desirable to have different protocols for different levels of difficulty of requests. In this way simple requests are easy to formulate, e.g., push a button, while more difficult requests require more effort.

RESPONSIVENESS

Ideally, the system should always respond immediately to a user's request. However, this is not always technically possible. In the absence of this ideal, the responsiveness of the system should be related to the nature of the interaction. Thus, if a request is short or very easy to formulate, then the user should expect that the system can respond to the request quickly. Conversely, if the request is long or very difficult to formulate, then it is not unreasonable to expect to wait longer for a reply. It is desirable to match difficulty in formulating a request with difficulty in processing the request. In this way, the user is not frustrated when he has to wait a long time for a reply to a request which he perceives as being very easy to formulate.

CONTROL

People usually like to feel that they are in control of a situation or at least to know that something they trust in is in control. They do not react favourably when they feel that they are being driven, especially if the driving force is a machine. In interacting with a computerized system, the user should always feel that he is the one that is in control and that the system is his friend. Even though the system is always in control ultimately, it should make the user feel he is in control. The responsiveness and helpfulness of the system are some determining factors that contribute to a feeling of control. The amount of interaction is also important. One feels much more in control in a car where one is constantly doing something to guide the car, than in a bus where one is a passive participant. Thus, receiving feedback from the system as one formulates a request can contribute to the feeling of control. This sense of control may be the most important factor in user acceptance of a system.

2.4 Summary

We have outlined several desirable characteristics of a query language. Sometimes the requirements for desirable characteristics mesh nicely. For example, bandwidth can be maximized by minimizing keystrokes per unit time. At other times the requirements are conflicting. For example, providing many protocols in a query language may sacrifice uniformity since different commands may be used in different protocols. The goal then in a query language design should be to determine and to evaluate the tradeoffs among the different requirements to come up with an appropriate choice of the user interaction parameters. Rather than choosing one value, it might be desirable to provide several versions of a query language in which different requirements are emphasized. Human factors evaluations can be helpful in deciding on appropriate values for parameters and appropriate approaches.

As an example of the evaluation of user interaction in terms of these requirements, consider requests for stock quotations or weather reports. One of the ways that we can provide the service is to have a complete stock or weather information report available on request. The person pushes a software button "stocks" or "weather" and the system responds with the standard stock or weather report in text form. Let us look at the user interaction parameters to evaluate the desirability of such a service.

In terms of formulating the request, keystrokes are minimal, the command is simple, bad requests cannot be formulated and this type of command can match many applications (by using different buttons) and can be used for many applications. However, the command has no selectivity at all. The user will always get a full stock or weather report. He can not isolate the business sector, company or time in terms of a stock report, or the geographic area or time in terms of a weather report. He gets everything and the onus is on him to pick out the relevant information from what is displayed.

In terms of getting a reply the presentation of the answer is fairly easy to read, the user can scroll the answer with any speed he wants and he can presumably save the reply for future reference. However, the user normally does not have a choice about via what medium or in what format he will get the information. Neither can he match the reply to his terminal's special capabilities, e.g., grahics if they exist. He gets the answer is textual form because it is the common denominator for all output devices. However, a child and a successful businessman may want different output for weather, i.e., the child in terms of a cartoon, while the businessman in terms of a terse factual report.

Consider now the dynamics of interaction. They are virtually non-existent. The request is formulated by pushing a button. This has very high bandwidth in terms of the speed with which the user's request is communicated to the system, but thereafter the interaction is over as far as the user is concerned. He has no control over the answer; he is completely passive. It is a very sterile and boring environment in which to select information.

3 QUERY LANGUAGE EVALUATION

In this section we categorize query languages into seven types. For each type we briefly describe the form of the languages. We then discuss the language type in terms of the user interaction parameters outlined in Section 2.

3.1 Keyword

Keyword query languages allow the user to use a restricted form of his native language to interact with the system. There are numerous examples of these types of languages among them SQL [Denny, 1977], ILL [Lacroix and Pirotte, 1977a,b] and NUL [Deheneffe and Hennebert, 1976]. Requests are formulated in a restricted, English-like grammar where specific words (keywords) signal the start of specific parts of a request. To make the processing of these languages easier, they are very rigid in their syntactic form. Constructs in the language must appear in a fixed order and in a fixed format. Replies to requests are usually presented to the user in a tabular format.

To formulate a request, quite a bit of keystrokes are required on the part of the user. However, once a request has been formulated it is quite easy to read and to interpret since the structure of these languages closely follows English sentence structure. There are usually only a few commands available to the user, but they are quite complex, allowing many options of specification. Thus, it normally takes some time to master an entire language completely although subsets of it may be learned quite quickly [Reisner *et al.*, 1975; Reisner, 1977]. Because of the complexity of each command and the amount of keystrokes required, it is not difficult to form syntactically incorrect requests. However, these are caught immediately by the system. It is possible to form semantically meaningless requests, but difficult if the request is syntactically correct and if the data base has been designed properly. The selectivity of keyword query languages is very high. A user can specify very precisely the data he wishes to select. The languages make no assumption about the nature of the data base in terms of application although they usually assume a certain overall structure to the data, e.g., relational. Thus the languages are application independent, but data base type specific. Keywords in a language can be customized for a given application. Other aspects of a language are rigidly fixed.

Replies to requests are usually presented to the user in a tabular format. The user has no control over the form of the output. Although tables are fairly easy to follow for some types of output, they can be very difficult to interpret if there are complex relationships between tables. Output is normally presented only on a screen or a printer. Customizing the output for the user is not provided for by the languages. Output is usually presented with table and column headings and the user may be able to customize these. No dynamic control over the output is provided other than that provided by the nature of the display device itself, e.g., scrolling. Most keyword languages will allow the user to save the result of a query and/or use it for further processing.

The dynamics of interaction for keyword query languages are very poor. The bandwidth of interaction is very low. Requests must be typed in and replies must be read from a screen or printer. There is virtually no gamesmanship involved in the interaction; everything is very cut and dry. There is only one protocol for interacting with the system: typing in the format required by the language. In some keyword query languages requests that are difficult to process, e.g., joins in relational systems, are quite simple to specify in the language. Thus the responsiveness of the system does not match the complexity of request formulation. Because of the limited way requests are formulated, the system is not able to provide much feedback to the user until after the entire request has been specified. In such an environment there is not much feeling of control over the system by the user.

3.2 By Example

By example query languages are ones in which the user constructs his requests by giving the system an example of a reply to the request. The example is constructed according to the format for replies used by the system. For example, in Query-by-example [Zloof, 1975a,b; Zloof, 1977, 1980] templates of tables are displayed on a screen and the user fills in the table according to which columns he wants in the reply and the kinds of values desired in each column. In TLA, templates of forms are filled in to specify a query [Hogg, 1981; Hogg *et al.*, 1981; Nierstrasz, 1981]. Similar languages have been proposed by other researchers [Hammer *et al.*, 1977; Luo and Yao, 1981].

In these types of query languages, while some input is required from the user, the keystrokes required are quite small. The system supplies a great deal of the input, such as attribute names, that would be required in a keyword query language. The user need only indicate which attribute values he wants in the reply (usually involving a single keystroke) and any conditions on the selection of values. Commands indicating the operation desired are usually limited to one character codes. The commands in these languages are few and simple. However, they are not well suited for specifying complex queries that span several templates. The readability of the request is compromised somewhat for simplicity of expression. It is very difficult to formulate syntactically incorrect requests because of the rigid form of the specification. For this reason semantically incorrect requests are also difficult to form. The user always is aware of the information the system has since it is displayed to him in the templates and only requests about this information can be formulated. The selectivity of these languages is comparable to that of keyword query languages. The way in which the queries are specified is independent of the application; only the templates change, not the manner in which they are filled in. As long as the data base that is being accessed can be represented by the form of the templates used in the language, then the language is also data base independent. Customizing of the language for an application can only be done in terms of attribute names within the application. Because of the rigid language format, it is difficult to take advantage of any special characteristics of an application.

Replies to requests are presented to the user in terms of the templates used to formulate the request. Thus there is a one to one correspondence between requests and replies. In addition, since these are usually in the form of tables or forms, they are fairly easy to follow and understand. However, this characteristic also limits the reply in that it must be presented on a screen or printer. In the same way that customizing of the requests is difficult, customizing of the replies is also difficult. Their form is determined by the form of the request. Because of the nature of the replies, dynamic control is not possible except for screen scrolling. These languages usually allow the user to save the result of a request and to use it for further processing.

In terms of dynamics, we have a request-reply environment in which the two processes are disjoint. The bandwidth is fairly high in terms of time of specifying requests and is related to the complexity of the request. Simple queries can be specified quickly, while more complex ones require some effort. The output bandwidth is limited by the speed at which a user can read a screen. These types of languages have a somewhat higher degree of gamesmanship than keyword query languages. Specification is less tedious and in some systems the user is guided by the system in filling a form or table, i.e., the cursor can skip automatically to the next attribute rather than the user having to space to it. On the other hand, the user is constrained to always specify requests by filling in templates. In terms of responsiveness, we have already mentioned that some queries, e.g., those that go across templates, are difficult to formulate. This difficulty of formulation also corresponds to those queries that are difficult to process. Because of the request-reply nature of the environment, the degree of control that the user has over the system is not high. Some guidance can be provided by the system, but the helpfulness of the system is limited.

3.3 Natural Language

Natural language query languages attempt to allow the user to specify requests in his native language. The goal is not to limit the vocabulary in any way, although in practice only a subset of a language can be processed by the system. There are many examples of such systems developed by researchers in artificial intelligence [Woods, 1973; Mylopoulos *et al.*, 1975, 1976; Harris, 1977; Hendrix *et al.*, 1978]. Perhaps the best known example that makes use of data base technology is the RENDEZVOUS system developed at IBM San Jose Research Labs [Codd *et al.*, 1978].

Requests to such systems are formulated in terms of a dialogue between the user and the system. This means a very high number of keystrokes are required to specify a request. There are no fixed commands *per se.* Like for English sentences, simple requests are fairly easy to specify while complex ones are difficult. Because the user has complete freedom to use the entire English language, the same request can be specified in many ways. This means that there are many ways in which a request can be formulated badly and misunderstood by the system [Shneiderman, 1978]. Syntactically, these systems usually provide spelling correction features. These languages can be very selective, but this depends on how precisely the user specifies his request. Such specification can be very difficult since the language provides no guidance in this matter. Natural language is the most uniform language one can have for specifying requests. It is uniform across applications as well as across data bases. In addition, the language can be customized by the user according to the application simply by using his knowledge about the application. This assumes that the system also has some knowledge about the application so that it can understand the vocabulary of the user.

Replies to requests can be presented in terms of English sentences for single facts. For many facts, this could pose some difficulty. Usually replies are presented in terms of typed sentences on screens. However, there is the possibility of adapting the sentences to voice output. The form of the output, as English sentences, allows for its customization. Words specific to an application or to the type of audience can be used. Because of the dialogue nature of the interaction, the user is able to exercise dynamic control over the reply to a request. The user can interact with the system until the reply contains the information of interest to the user. Reusing and saving the replies is difficult unless the system has the capability of digesting new facts that are presented as natural language sentences.

The dynamics of interaction consist of a dialogue between the user and the system. For typed input, this has a very low bandwidth since entire sentences must be typed. If the output is also in terms of typed sentences, then this bandwidth is also very low. Since the user is carrying on a dialogue with the system, the system can incorporate some gamesmanship into the interaction by the way in which it responds to the users query. For example, it can ask the user to clarify parts of his request that it does not understand. It can also rephrase the user's request in different terms. This also allows multiple ways for the user to interact with the system, e.g., by stating sentences, by answering the systems questions or by making choices supplied by the system. Because requests are formulated in a piecewise fashion, the user always has the illusion that he is making progress in satisfying his request. This also allows the system to process complex requests in a piecewise way so that the delay between request formulation and response is minimized. The piecewise approach to request formulation also makes the user feel that he is in control since he is guiding the system to the answer.

3.4 Menu

Query languages that are menu-based allow the user to specify requests by pointing at the action or data that is desired. At any point in time, the user can choose from a menu of available operations or data objects. Choosing an operation or object may cause additional menus to be made available. In this way, requests are specified by steps in a hierarchical manner. These types of systems have been used extensively in many applications. The Officetalk system at Xerox Palo Alto Research Center is an example of a system that uses menus for querying data bases [Ellis and Nutt, 1980].

Requests are specified by pointing at desired actions. Thus, a minimum number of keystrokes are used. The commands available are simple and represented by either English keywords or by icons that represent the action or object of interest. For example, to store a document on the screen one can point at the document and then at the image of a filing cabinet. Because of the hierarchical structure of menus, the commands are adequate for simple requests that are hierarchically structured, but inadequate for more complex requests. The highly structured nature of the interaction makes it difficult to formulate incorrect requests. The interaction is completely under the control of the system at every stage. Selectivity is very good for queries that have been anticipated, i.e., those that follow the menu structure, but more difficult for other requests. The same technique of menu selection can be used across applications and even across data base types. Menus can be customized to specific applications by changing the keywords or icons that are used.

Replies to requests are presented to the user in the form of the objects that are stored by the system. In most systems employing menu techniques these are usually forms containing structured text or a table. Thus replies presented in this way are fairly straightforward to follow. Output is normally available only on a screen or a printer and must be read by the user. If the system allows definition of new objects on the fly as in Officetalk, then the output can be customized for the user. However, this customization is only in terms of the types of objects that are maintained by the system, e.g., forms. The user can dynamically control the presentation of the output if he is allowed to use the menu operations as the output is being presented to him. Thus, if a reply has several forms as the answer, then he may be able to look at each one in turn and take some further action upon it before looking at the next one. Facilities may be available for saving the result of a request and/or using it for further processing.

The dynamics of the interaction in these types of languages are somewhat better than in previous languages. Input bandwidth is reasonably good once the mechanics of the pointing operation have been mastered. This bandwidth increases if screens that allow multiple windows are used. Output bandwidth is somewhat restricted because of the necessity for reading the reply. The pointing type of environment has interesting possibilities for gamesmanship. It is certainly much less tedious than keyboard interaction and the use of icons to represent actions can help maintain user interest. In the Officetalk system the user can interact with the system in different ways. For example, he can switch to an editor to edit input and output, or to a drawing mode for free-hand sketches. Because the pointing operation on the part of the user has high bandwidth, the user may expect that all operations are equally complex. However, this is not usually the case. Thus processing speeds and response time may not be uniform which may frustrate the user. The nature of the interaction and its high bandwidth give the user the illusion that he is in control, always directing the system. However, this may not be the case for queries that do not follow the structure of the menus.

3.5 Graphic

Graphic query languages formulate requests mainly in terms of diagrams, but some typed input is also usually required. Each type of object in the data base is assigned a specific geometric shape. The user then manipulates these shapes to specify requests. Special conventions are used to specify selection of specific data or relationships between data. Two examples of these types of languages are the LSL graphic language [Lipson and Lapczak, 1976] and the CUPID language of the INGRES system [McDonald and Stonebraker, 1975].

Although keystrokes are minimal in formulating a request, quite a bit of work is required by the user to construct the diagrams that represent a query. Geometric objects have to be manipulated on the screen and processing options specified for some of them. Once the form of the request has been specified by a diagram, the action desired is fairly easy to invoke. The commands therefore are simple; the complexity is in constructing the diagrams. Syntactically incorrect requests are difficult to formulate since the way in which objects can be put together and connected is controlled by the system. Similarly, semantically incorrect requests are difficult to formulate, although it is possible. Graphic query languages can have the same degree of selectivity as keyword languages since they employ the same types of constructs only specified differently. The languages can be used across applications that use the same type of data base, since the data bases employ the same types of objects for representing data. Across different data base types, these languages may not apply unless conventions concerning the interpretation of geometric objects in different contexts are specified. Customizing for applications can be done in terms of the data object names used and keywords employed in the language.

Replies to requests are normally presented in terms of a tabular structure much like for keyword query languages. Therefore, replies have much the same characteristics in terms of presentation, multi-media, customizing, dynamic control and reusability.

The dynamics of interaction for graphic query languages are not particularly good. As a communications medium, pictures have a high bandwidth. However, the bandwidth achieved in formulating requests is not very high. It takes much time and effort to formulate requests via diagrams. It may be easier to specify the request by some other mechanism and use diagrams to display the specification, i.e., the system constructs the diagram, not the user. Diagrams would appear to have a high degree of gamesmanship. It is fun to draw pictures, but only if it can be done easily. Currently this is not the case with these languages, particularly for complex requests. In addition, it may not always be clear how to formulate a request in terms of a diagram, especially for naive users. Formulating requests graphically may be a long process even for simple queries. Thus the user gets no real sense of how complex a query is from its formulation. Also in terms of perception, the process of specifying requests is slow and will seem slow to the user because of limited interaction with the system. It is doubtful that the users of these languages feel that they are in control of the interaction.

3.6 Multi-media

Multi-media query languages, as their name implies, use many different ways of interacting with the system. Text, formatted data, voice, graphics, colour, animation, etc. are used to formulate requests and to present replies. Categories of data in the data base are represented by icons. This categorization can be specified at several levels of abstraction. Users can browse through the icons at a particular level and select one for further processing. The effect of icon selection is to "zoom in" on the data in question and to be presented either with further icons containing more detail about the category of data, or with the actual data itself. Icons can represent textual data, formatted data, visual data, etc. The most advanced work of this type is the spatial data management system developed at MIT and CCA in Boston [Herot, 1980].

Requests can potentially be formulated in many ways, e.g., by pointing at icons, by typing statements, by selecting from menus, etc. The same data can be selected in different ways depending on the mood of the user or his knowledge level about the data base. The commands used can be as simple as pointing or as complex as keyword query language specification. Because of the flexibility involved in formulating requests, there is the possibility of looking in the wrong place for data. This is especially true if one is in a browsing mode. The degree of selectivity possible depends to a large extent on the construction of the data base. That is, if there are many levels of abstraction between the highest level and the data, then the "zooming in" process can be fairly selective. Otherwise, a great deal of irrelevant data may be selected. Of course, the user has the possibility of using a more selective query language at any time. These techniques can be used across applications and across data base types. However, the particular technique used, e.g., pointing, may be more appropriate in some situations than in others. Customizing for applications is done in terms of the type of data that is stored about an application, and how it is stored, at each level in the abstraction of a data base.

Replies to requests depend on the level of abstraction one is at and the nature of the data base. It can be in terms of text, pictures, graphics, etc., whichever is the most appropriate representation for the data. This allows data to be stored in a way that is appropriate for the intended audience. The user can direct the system to display the reply in a certain way or via a certain medium. The user has complete control over how the output is displayed, where, in what form, and at what rate. Although the results of a query can be saved, it is not always possible to use it as input to another query, e.g., if it is in graphical form or verbal form.

In terms of dynamics of interaction, such a system is excellent. Input and output bandwidth is high and the user can choose the form depending on his tastes. The interaction has a high degree of gamesmanship. The user can "fly" over the data and zoom in on a particular part very easily. Interaction with the system can be done in several ways and can change as a request is formulated. Response time of the user can be very fast to a request by the system. However, the system's response time to a user's request may be quite slow because of the interaction of different technologies involved in the interface, at least at the present time. Thus the interaction may become frustrating to the user especially if response time is not uniform for similar requests. The perception of control by the user can be quite high in such a system. The user has available many ways of directing and instructing the system. The system can be helpful in guiding the user interaction by monitoring the nature of the interaction. For example, it can provide more help if the user is making many mistakes, or it can provide fewer levels of abstraction if the user is following a well worn, to him, path.

3.7 Computer Games

There have been many computer games developed for accessing useful information or for amusing researchers. Most of these games are not generally known because they operate on very specialized data bases in localized areas. For example, there is a game at Bell Labs called EATS for finding a restaurant of a certain type in a certain location. The idea behind these games is that you search for something by playing a game with the system. For every input you supply, the system tells you something about the nature of your current environment. From this one can tell whether one is near or far from one's objective. Techniques used for interacting with such a game are for the system to ask questions of the user or for it to supply the user with some choices for actions.

Requests in such an environment are specified incrementally. For example, in a question-answer type of game, the system asks a question, the user supplies an answer and the system tells the user something about the current status of the game. This type of request formulation can involve several keystrokes in supplying answers. However, answers are usually short and may merely involve a yes-no response. The commands are very simple and once initiated, the system guides the user so that it is impossible to formulate syntactically incorrect requests. However, semantically one can get into difficulty because responses to questions are made in the context of limited information about the current environment. Thus one could easily head off in the wrong direction. If one keeps a straight path to the desired information, then the selectivity is good. However, if one strays, then a great deal of useless information may be obtained. Unfortunately, the nature of the request formulation is very application dependent. On the other hand, the techniques used can be very insensitive to the type of underlying data base. The interface can be easily customized to particular applications.

Replies are presented to the user in a piecewise manner. Usually parts of the reply are presented to or discovered by the user as the request formulation proceeds. Thus, the presentation is usually very easy to follow since one is presented with only a few facts at a time. Replies are usually in terms of screen output, but sound can be used to indicate that one is nearing an objective or is on the right track, e.g., as in T.V. computer games. The form of the response can be customized to the intended audience as in T.V. computer games. The user can control the rate at which he obtains the output by controlling the rate at which he responds to the system. Because of the nature of the interaction, saving previous replies and reusing them can be difficult.

In terms of dynamics, the interaction can be a lot of fun for the user. Input and output bandwidth can be fairly high especially if input requirements are minimal, e.g., yes-no responses. The interaction obviously has a great deal of gamesmanship in it. The user is playing a game in which he is trying to win something. For each game there is normally only a single protocol of interaction, but this protocol can be different for different games. Because of the step-wise, short interaction, response of both the system and the user to each other can be quite fast. System response should be uniform since at each step only a relatively small number of possiblities arise (especially if yes/no answers are expected). The perception of who is in control of the interaction depends on the users perception of whether or not he is winning. If the game is designed properly so that it is not too difficult to win, then the user will feel that he is in control.

4 QUERYING REQUIREMENTS

In a previous section, we divided the querying process into three parts: request, reply and dynamics of interaction. As far as the user interaction is concerned, the most difficult of these three parts is the request part and its dynamics. The reason for this is that in forming a request the user is trying to inform the system precisely of what he wants. The user must provide the system with a great deal of guidance and be very exact in specifying a request if the system is to understand it correctly. Replies from the system on the other hand are much easier for the user to understand even if they are presented badly. This is because humans are much more intuitive and adaptable than computer systems. This is not to say that reply aspects of querying are not important. However, for the rest of this report we will concentrate mainly on the request aspects of a query language and its dynamics.

For both the request and dynamics aspects of a query language, certain characteristics were identified that described the parameters of the user interaction. From these characteristics and their evaluation with respect to different query languages, we can derive certain desirable characteristics for a query language for external data bases.

In general, the user wishing to access the external data bases is not a typist and probably does not want to become one to use the query language. Hence, there should be a minimum number of keystrokes required to communicate with the system. In order to meet this requirement it is essential that some type of pointing mechanism be available for pointing at things on the screen. The pointing method should be mechanical, e.g., a joystick or mouse, so that the user is free to concentrate on the screen while manipulating the pointing device. Most likely, it will be impossible to eliminate all typing. Therefore, to be least tedious, it is also essential that typing mistakes be allowed and corrected by the system and that the user be able to assign aliases to refer to objects in the external data bases.

Any commands that are required to direct the system to perform some action should be provided via function keys or menus of operations and be named to reflect their functionality. In addition, the number of such commands should be kept to a minimum. In this way the user is not required to remember the commands available and can easily recall their function from their names.

The system should guide the user in formulating his requests. To this end it should display options available and also provide instructions on-line at any point during an interaction. The way in which requests are formulated should be intuitive to the user. In this respect, there should be a paradigm for formulating requests that the user can easily understand and that helps him remember the interaction protocols. In addition, the system can provide cues to the user, e.g., by the form of the display, that help the user formulate correct requests.

While providing guidance to the user in formulating his requests, we must be careful not to put the user in a straightjacket by completely predetermining the requests that can be formulated. The query language must still allow the user some latitude in determining the selectivity of his requests. To this end, the query language must at least allow the user to specify the contents of his reply, if not its exact format.

In most query languages the user is supposed to know the structure, called the schema, of the data base he is accessing. For external data bases, we cannot expect the user to know or even understand the structure of the data base or the difference between the schema and the data base. Instead, there should be a mechanism to guide the user through the structure part to the data part. In addition, the way in which the user queries the structure of a data base should be as similar as possible to the way he queries the data base itself. In this way the user only needs to learn one set of interaction protocols that apply to all his interactions with the system.

Studies have shown that users of computer systems can become very lost if they do not know where they are in the system [Mantei, 1981]. In addition, a user cannot be expected to remember everything that has transpired during an interaction. To help orient the user, we should retain on the screen as much as possible the route we have followed in querying the data base (or at least have it available for display).

User interfaces for querying nonformatted data are often very different from those that query formatted data. When querying external data bases, we may have many different types of data available. Most data will probably be in such a form that a user cannot distinguish formatted data, e.g., inventory information, from nonformatted data, e.g., catalogue sales information. In addition, it is probably not desirable to require a different set of protocols for different types of data. Thus, the user interface should be as uniform as possible for querying the different types of data found in external data bases.

Even though we provide uniform interaction protocols for all types of requests, we can allow a very nonlinear user interface in terms of difficulty of request formulation. That is, simple requests are extremely easy to formulate while more elaborate requests can be very complex. In fact, joins and other complex requests can be assumed not to be frequent. In any case, we will suffer in performance when we do them. The user interface should allow these queries through a higher level, expertise interface which can access the same data as the interface for naive users.

Finally, the user interface should use to advantage the ability of people to remember abstract patterns. In addition, the available technology (graphics and colour) should be used to distinguish between different types of things on the screen. For example, we should distinguish between the data that the user is searching for, is providing for selection or is provided by the system. Although we will not elaborate further on this aspect in this report, it can be very important for guiding the user in his interactions with the system and should be carefully considered in the design of the screen layout presented to the user.

With these considerations as our guiding principals for request formulation and user interaction, we will outline a design for a query language for external data bases in the rest of this report. We first present a paradigm for the query language that conveys by an easy to understand example the essence of the user interaction. We then present an overview of the query language design. Finally, we discuss the design considerations for the query language.

5 PARADIGM

For accessing information via an interactive query language for external data bases the user should have a paradigm that he or she can follow and easily understand. This paradigm should help the user remember (and learn) the querying protocols. The paradigm that we choose is that of navigating a personal spaceship through space.

Our spaceship is of the latest design and is very powerful. It can travel both long distances, e.g., those between galaxies, and short distances, e.g., those between nearby planets, with relative ease. Our usual objective in travelling through space is to arrive at a planet and explore it, although we may just want to explore space per se. Once we arrive at a planet, we can use our spaceship to explore it by flying around or by landing and examining an area in more detail.

Space is very vast and complex, and usually unfamiliar to us. Our spaceship is equipped with a view screen which can show us a visual picture of space in any direction, but only a limited portion of it at a time. It is very difficult, in general, to navigate without aids, i.e., visually only, although when we are exploring an area of a planet this may be desirable. To guide us in our travels, our spaceship can provide us with detailed maps. These maps come in different levels of abstraction from those showing the general structure of a galaxy to those showing the general structure of a planet. These maps can be displayed on the view screen in the spaceship.

We can navigate our spaceship in one of several ways. If we know where we want to go, we can specify the destination and our spaceship can automatically pilot us to the destination. On the other hand, if we are just exploring space, then we may not know our next destination specifically. We may have some idea of the type of destination we want to arrive at, or we may just be interested in looking around space. To learn something about the potential destinations we can look at the maps of them on the view screen. We can ask to look at either specific maps having a certain property or properties, or at all maps. In general, as we explore possible destinations we will look at several maps from very general ones to more specific ones, e.g., from galaxy to star system to planet. Eventually, we will choose a destination and direct our spaceship to take us to it.

In a similar fashion, we can think of the information sources that we wish to access as residing in a vast *information space.* The composition of this information space is patterned after the composition of space itself. It is composed of abstract objects, such as galaxies and star systems, and concrete objects such as planets. In the case of the information space, the abstract objects correspond to the *structure part* (schema) of the data. The structure part tells us something about the nature of the data we can access. The concrete part corresponds to the *data part* (data base) of the information space. The data part is the smallest level of detail that we can explore in the information space.

Using our spaceship (the computer system) we can explore the information space. Our spaceship command post in this case is provided by the query language and our view screen is provided by the CRT (TV) screen through which we view the information space. Just as we can only see a portion of space at a time on our spaceship view screen, so we can only see a portion of the information space at a time on our CRT screen. The CRT screen defines the boundaries of the information space that are currently in view as our spaceship view screen defines the part of space that can currently be viewed. We will call the CRT screen our *portal* on the information space since it acts like a view port allowing us to see only that information currently in view through the view port.

In order to manipulate and maneuver in the information space we require three basic actions, much like for maneuvering through space. First, we need to be able to display in our portal the appropriate structure parts (maps) of interest for the information space so that we can select our destination. Second, we need to be able to go to our destination, that is gain access to the information source of interest. Finally, we need to be able to explore the data in the selected information source.

We note two points about this paradigm. First, we distinctly separate the structure part from the data part conceptually, but operationally (as we will see) they are handled (almost) the same. This separation could be important in the future as the structure part could reside in the user's terminal much like the maps in the paradigm reside in the spaceship. The data part exists at an information source and to access and explore it the user has to "go to" the information source just as he has to go to a planet to explore it in detail. Second, the way in which we have described the user interaction with the system, most of the details of how to access information are automatic. The user supplies certain guidance, but the system worries about the details of how to do things. This mode of operation is preferred for the novice user, but may be tedious for the experienced user. Therefore, we could provide a manual override to the user and let him guide the system more directly, much like we could provide a manual override in our spaceship. This corresponds to the option of providing a more powerful and complex query language for the experienced user. In this report we only discuss the "automatic" query language, but the possibility of a "manual" query language as a complement to the former should not be discounted.

6 DESIGN OVERVIEW

In this section we will describe the overall design of an interactive query language for accessing external data bases. We will discuss *what* can be done and not *how* to do it. Thus we will describe the operations generically and use generic terms such as "point at", "fill in" and "select" without discussing how these operations are done. We will use the analogy developed in the previous section to explain the operations in the query language. In the next section, we will consider some of the ways in which these operations might be done by the user.

6.1 Choosing an External Data Base

Suppose that we are in our personal spaceship somewhere out in space and would like to choose a destination to go to. Using our view screen, we can scan space and look at the stars. However, all that we see by doing this is an abstract pattern. If we are experienced space pilots, we may be able to choose a destination from the star pattern directly and direct our spaceship to go to it. However, in general we probably will not be familiar with space. To choose a destination for our spaceship, we can scan different parts of space and look at maps of each part. We can ask to look at only certain parts of space that have some specified properties or we can look at all parts of space. As we look at the maps, we can ask for a more detailed map of a specific region of the current map. While looking at the maps, we may mark some regions of a map for future reference as being of interest and requiring a second look. Eventually, we will decide on a destination and direct our spaceship to go there.

Consider now trying to access external data bases. When we first sign on, we are like the person in the spaceship. We are out in space and in the portal we can see an abstract pattern (template) of the data bases that we can select from. The image that we see is like the view of space that just shows us the star patterns without any information about what they represent. For example, we can be presented with a template in the portal such as that shown in figure 1.

The exact format of the template in the portal is not important here (this is subject to study to determine the best layout). The important point to note is that the template is divided into three basic

**** VIDEOTEX INFORMATION SERVICE ****

ROUTE:____

CATEGORY	CONTENTS

MESSAGE:____

Fig. 1 Example template.

**** VIDEOTEX INFORMATION SERVICE ****

ROUTE:____

CATEGORY	**CONTENTS**
Entertainment	Films
	Live arts
	Exhibits
	Sports
	Television
News	World
	National
	Local
	Sports
	Entertainment
	Weather
Restaurants	Chinese

MESSAGE: Frame 1 of 10

Fig. 2 Example display.

parts. One part specifies the *path* we have taken to arrive at the current template in the portal. The **ROUTE** field in the example contains this information. A second part is used for the *specification and display* of data. In the example the **CATEGORY** and **CONTENTS** fields serve this purpose. Finally, there is a part used for *system feedback* to the user, labelled **MESSAGE** in the example.

If we have no idea which external data base we would like to access, we can ask the system to display all the information it has about data bases at this level. In this case we get a display of all entries that correspond to the template. For example, we might get a display such as that shown in figure 2 if we initially ask for a display of all entries.

We now can examine the information map, marking some entries for future reference by pointing at them or selecting one by pointing at it. Selecting an entry allows us to look at more detailed information about it. For example, selecting the category 'entertainment' and requesting a display of all its entries would give us a display such as that shown in figure 3.

Alternatively, we may know the kinds of external data bases we are interested in, but not where

**** VIDEOTEX INFORMATION SERVICE ****

ROUTE: Entertainment

CATEGORY	**CONTENTS**
Films	First run
	Revues
	Festivals
Live arts	Theatre
	Ballet
	Opera
	Recitals
Exhibits	Art
	Books
	Crafts
Sports	Hockey
	Soccer

MESSAGE: Frame 1 of 2

Fig. 3 Example display.

**** VIDEOTEX INFORMATION SERVICE ****

ROUTE: ________________________________

CATEGORY	**CONTENTS**
Entertainment	Films
	Live arts
	Exhibits
	Sports
	Television

MESSAGE: Frame 1 of 1

Fig. 4 Example display.

they are. In this case, we can ask the system to show us more information about these destinations by pointing at and filling in the **CATEGORY** field. For example, we may fill in the value 'entertainment' in the **CATEGORY** field and ask for a display of its contents as in figure 4.

Subsequently we can examine the information map, mark some entries for future reference and/or select one by pointing at it. For example, if we select the 'entertainment' entry and subsequently ask for a display of its entries we get the display shown in figure 3.

Rather than explicitly entering a field value, we can fill in a field value by pointing at the field and requesting a display of all the field values. We can then mark some entries as being the values we want to fill in for this field. In this case rather than entering a value explicitly, we do it implicitly by marking some values from those displayed. Thus, for example, we could specify the display shown in figure 4 by pointing at and requesting a display of the values in the **CATEGORY** field, marking the entry 'entertainment' and requesting a display again.

Finally, if we know where we want to go, then we can specify this directly and the system will take us there. We specify this information by pointing at and filling in a value for the **ROUTE** field. This directs the system to a certain destination and causes it to show us a template of the destination selected.

By applying this technique repeatedly, we can navigate the structure of a data base to arrive at the data of interest. Every time we select something in a template we get more detailed information about the route to a data base. We also retain information about the path we chose to get to the current template. This is analogous to trying to determine a destination for our spaceship. This technique can be carried out to several levels, but it is desirable to keep the number of levels small to reduce the amount of interaction required to get to the data. In general, the number of levels required will be determined by the complexity of the data base being accessed. This aspect is in line with our stated goal of relating complexity of access to complexity of the information or type of request formulated.

By means of the **CATEGORY** and **CONTENT** fields, we in effect provide a keyword and cross referencing capability. That is, all data bases related to a specified category or categories will be displayed. As well, some information may apply to more than one category. Note that the displayed information may be at different levels in the structure part. That is, some may refer to data bases themselves, while others may refer to parts of a data base. However, this is irrelevant to a user and he need not be aware of this distinction. What appears in the portal is a function of how the cross references to data have been designed within the system.

Associated with each keyword displayed in the portal is a more detailed description of the information represented by the keyword. At any point in time we can point at a keyword and ask for a description of it. This is analagous to consulting a guidebook associated with a map. We only do so when we want to know more detailed information about a point of interest.

The **ROUTE** field allows us to see the path we have taken to arrive at our current location. At any time we can backtrack by pointing at the level (name in the path) we want to go back to.

6.2 Exploring the Data

Our eventual destination in our spaceship is a planet. Once we arrive at a planet, we can fly around and look at it or we can land and explore a small area of the planet. Again to help us navigate, we require maps of the planet. These maps show us the abstract structure of the planet but no detail about it. By flying around and/or landing we can explore the planet in detail.

In a similar manner, we eventually reach the data part of a data base. As in exploring and navigating around the structure part, we can now explore and navigate around the data part. As for the structure part, we again are presented with an abstract pattern (template) of the data in the portal. An example of such a template is shown in figure 5.

**** VIDEOTEX INFORMATION SERVICE ****

ROUTE: Entertainment.Films.First run

FILM	**STARRING**	**THEATRE**	**TIMES**	**PHONE#**
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________
________	________	________	________	________

MESSAGE:____

Fig. 5 Example template.

Specific template entries are selected by pointing at and filling in a field. For example, to display all films starring Donald Sutherland, we would point at the **STARRING** field and enter the value 'Donald Sutherland'. The system would then display all films starring Donald Sutherland that are showing currently. Alternatively, we could fill in a field by pointing at it and asking the system to display all values that it has for that field. We can then mark certain field values by pointing at them. The system will then show us a complete display of all entries in which that field value appears. For example, we could ask that the system show us all values for the **STARRING** field. We then choose those values of interest to us for further inspection.

If we are just interested in browsing, then we can ask that all template entries be displayed. Subsequently, we can mark specific entries for further examination by pointing at them. This action has the effect of eliminating those entries that are not of interest to us and thus reducing the size of the data display.

6.3 Predefined Routes

As we travel around in our spaceship, we soon discover that there are certain destinations that we go to very often. We become familiar with the routing to these destinations and soon do not require maps to help us get there. We can specify the destination precisely by visual navigation.

In a similar manner, when accessing external data bases there will be certain data that we access very frequently. To speed up our routing to this data, we can predefine the route that the system should take. We can name this route and then specify it explicitly in the **ROUTE** field. This is analagous to pointing to our destination directly on the visual star patterns that we see from our spaceship.

These predefined routes are appended to the structure part of the videotex information service, but are only available to the user who defined the routes. The destination of the routing can be a point in the structure part, data part or part of the data part, i.e., a subset of the data. In this way the output that the user receives can be customized to some extent by, for example, filtering out certain information in the data part. Predefined route names can be distinguished from regular data base names by highlighting them in some suitable manner.

7 DESIGN CONSIDERATIONS

In the previous section we discussed what the query language can do, but no details were provided about how it might do it. We used generic terms such as "point at", "fill in" and "select" without specifying how these actions might be done. In this section we will elaborate on how some of the actions discussed in the previous section might be done by a videotex user.

7.1 Pointing

One of most fundamental actions required in the interaction protocols discussed in the previous section is the ability to point at something within a portal for the purpose of marking or selecting it for further actions. Ideally, this pointing should not distract the user's attention from what is being displayed in the portal. Voice directed pointing would be highly desirable, but perhaps not technically or economically feasible currently. The next best pointing mechanism would be a mechanical pointing device such as a joystick, puck or mouse. The user is then able to position a cursor to the appropriate position within the portal and to indicate the selection of this position in some way. (Note that it is neccesary to both position the cursor and to indicate that this is the position desired.) Such a pointing capability is technically feasible now and possibly could be provided economically.

In the absence of a pointing device for specifying an arbitrary location in the portal, an alternative is to be able to specify specific locations, namely specific fields in the portal. This specification should be made as easily as possible and should take account of the fact that positioning is a function of the template structure currently displayed in the portal. We generally need to be able to specify a field in a template and perhaps an entry within a field. For generality, these two specifications should be independent of each other. The exact semantics of the specification will depend on the characteristics of the template displayed in the portal.

The ability to select a specific field and a specific entry leads to the notion of a *current* field and entry. Visually this situation can be indicated by highlighting the current entry and field in some way, e.g., blinking, inverse video or a certain colour. Operations can be specified that then apply to the current entry or field as discussed later in this section.

For field specification, there is a **field** operation which has the following options and semantics: **field** *next* — selects the next field **field** *previous* — selects the previous field **field** *name* — selects the named field.

Fields in a template have a predefined ordering which is used by the *next* and *previous* options. Fields also wrap around. The initial field is the entire template as is the field after the last field and before the first field. The *name* option uses a symbolic name to refer to a field directly. Conventions need to be established for naming fields symbolically. Some functions (as defined below) require a field as a parameter. If no field is explicitly specified, then the default field is the entire template. Thus, **display** mode with no field explicitly specified refers to a display of all the values associated with a template, while **display** mode following the selection of a field refers to a display of only the values associated with the selected field.

For entry specification, there is an **entry** operation that allows an entry within a field to be selected with the following options and semantics: **entry** *next* — selects the next entry **entry** *previous* — selects the previous entry **entry** *name* — selects the named entry.
Entries have a predefined ordering which the *next* and *previous* options utilize. Entries can also be referred to by symbolic names which allows an entry to be selected directly. For example, for a list of

values within a field, entries can be referred to symbolically as 1, 2, etc. If the current field is the template, then an entry refers to all the values that define one instance of the template.

7.2 Portal Positioning

In our discussion so far, we have not considered any restrictions on the size of the template that could be displayed in the portal. We implicitly assumed that everything could be displayed at once. In general, however, templates may be larger than the portal size. Thus, the size of our portal limits the amount of information that we can display at any one time. Ideally, we would like to be able to move the portal continuously across the template as if it were a TV camera.

In videotex systems, the information space is usually divided into fixed size units (pages) which are the size of the portal (TV screen) and continuous motion in all directions is not possible. Accordingly, we can assume instead that templates are divided up into fixed size *frames.* A frame is a part of the information space that can be viewed through the portal at one time (much as one views a photographic slide through a slide projector). In general, frames exist in all directions from the current frame (rather than just forward and backwards as when viewed through a slide projector).

A frame contains just that amount of data that will fit entirely within the portal. Frames above and below the current frame hold, respectively, previous and next template entries. Frames to the right and left of the current frame hold additional data related to the current frame that cannot be displayed all at once through the portal. Thus, for example, tables of data that are too wide to fit within the portal can be divided up into several vertically adjacent frames (right and/or left of each other). While tables of data that are too long to fit within the portal can be divided up into several horizontally adjacent frames (above and/or below each other).

To position the portal to the desired frame, there is a **frame** operation with the following options and semantics: **frame** *up* — moves the portal to the frame above the current frame **frame** *down* — moves the portal to the frame below the current frame **frame** *left* — moves the portal to the left adjacent frame **frame** *right* — moves the portal to the right adjacent frame **frame** *name* — moves the portal to the named frame.

Frames can be named symbolically so that it is possible to position the portal on them directly. In most cases, and probably initially, it is desirable not to have right and left frames as this will fragment the data a great deal. In this case the **frame** function reduces to the *up, down* and *name* options.

If it is necessary to have left and right frames, then some way of orienting the user as to his current position is desirable. This orientation has two aspects to it. First, we want to tell the user where he is with respect to the entire information space visible at this level. One technique used to do this is to graphically show the position of the portal with respect to the entire information space. For example, we could show the entire information space as a polygon and the portal position as a polygon superimposed on the larger polygon representing the information space. Alternatively, we could use the message area to indicate the displacement of the user left and/or right of some reference point much as we showed the displacement vertically in previous examples.

The second aspect that we need to worry about is maintaining a semantic connection between the different frames in the portal. In the vertical direction this is not a great problem since as we scan up and down the same part of the template is displayed in the portal. However, in the horizontal direction the part of the template visible in the portal changes as we move left and right. It is therefore desirable to keep some common thread between horizontally adjacent frames. One possibility is to choose one of the template fields as a *key* field and to keep this field in the portal at all times as we move horizontally.

For example, in figure 5 one of the fields of the template, say FILM, could be kept in the portal as we move the portal position horizontally.

7.3 Control Functions

Control functions deal with signalling the system that it should note something about the current state of the system and perform some action based on the state. The effect of invoking a control function can be thought of as invoking a procedure that takes as its input the current system state and/or field value. Depending on the value of the input parameter(s), certain actions occur. Control functions can be provided in one of several ways, e.g., as function buttons or menu commands.

SELECT

The **select** function takes as its input parameter the current field value. The result of the function depends on the characteristics of the field value. If we select a category field value, then the result is to make that category part of the route we are following. Selecting a name in the **ROUTE** field causes the system to back up our routing to that point. As we will see, selecting a data value that is of type *image* or *voice* results in the display of a picture, the showing of a film or videotape, or the playback of recorded voice messages. If a field value has no explicit procedure defined for it, then selecting it has no effect, i.e., a null action results.

MARK

The **mark** function takes as its input parameter the current field value. The result of the function is that the system notes the marked value for future reference. For example, when we are examining field values we can mark certain ones for further action. Marking can only be done in **display** specification mode.

DESCRIBE

The **describe** function takes as its input parameter the current field value. The result of the function is that the system provides a detailed explanation of the current field value. This explanation can be visual, verbal or both. For example, for a category field value the **describe** function can be used to get an explanation of the information available in the category. Not all field values need have descriptions associated with them. Therefore, invoking this function can result in no description, i.e., a null action.

HELP

The **help** function takes as its input parameter the current system state. The result of invoking the **help** function is to receive instructions as to what actions can be performed by the user, and how, given the current system state.

CANCEL

The **cancel** function takes as its input parameter the current system state. The result of the function is to undo the last control function or the current specification mode.

SUSPEND

The **suspend** function takes as its input parameter the current system state. The result of the function is to save the current state of the user interaction. A suspended user interaction can be reactivated at any time. The interaction proceeds from the point of suspension as if it had never been interrupted. Suspended user interactions are kept in the data base named 'suspended' and can be reactivated by selecting them.

EXIT

The **exit** function takes as its input parameter the current system state. The result of the function is to end the current user interaction. No information about the current interaction is saved.

SAVE

The **save** function takes as its input parameters the current system state plus a user supplied name. The result of the function is to save the specification of the user interaction and to name it explicitly. The interaction can subsequently be duplicated by selecting the user defined name. The user defined name becomes part of the structure part information for this user. The **save** function provides a view definition capability. The **save** function differs from the **suspend** function in that a suspended interaction is only maintained temporarily until it is completed. A saved interaction on the other hand is maintained permanently by the system. A saved interaction can be dropped by saving a null (initial) system state with the same name as a currently saved system state.

7.4 Specification Modes

Specification modes deal with the way in which a user specifies requests for information to the system. Invoking a specification mode implies that a sequence of actions will follow to form the specification. A specification mode is ended implicitly by the **select** function, **save** function or another specification mode, or explicitly by the **cancel** function or **exit** function. When interacting with external data bases, there are two ways that the user can specify his request. One of these will be the default mode; which one is determined by the user. The default mode is entered initially when the user signs on and after every **select** function or **save** function.

FILL

Fill mode allows the user to select various fields and to specify a value or values for each field selected. Fields are selected by pointing at them. They are filled in explicitly by entering a value from a keyboard or implicitly by marking displayed values as explained under **display** mode.

If multiple values are specified for a field, then the *or* boolean operation is assumed among the values, i.e., match where value-1 or value-2, etc. Note that an *and* boolean among values does not make sense unless an entry within a field in fact can have multiple values. If more than one field is filled in, then the *and* boolean operation is assumed among fields, i.e., match where field-1 and field-2, etc. An *or* boolean among fields can be handled by filling multiple templates.

The end of template filling is signalled by entering **display** mode with the current field being the template. The user is then shown all entries of the template that match the specification. If no fields have been filled in, then all entries corresponding to the template are displayed.

DISPLAY

Display mode allows the user to look at entries of fields or templates in the portal. If the template is empty when **display** mode is invoked, then all entries are displayed. If the template has been filled in, then only entries that match the filled in template are displayed as discussed under **fill** specification mode.

In **display** mode the template entries can be scanned using the pointing and portal positioning operations. The current field and entry can be selected using the **select** function which invokes the procedure associated with the current field value. Field values can also be marked in **display** mode. Subsequently, entering **display** mode again will select only marked field values for display. If the current field is the template, then marking has the effect of selecting only the marked entries for display. If the current field is a field within the template, then the effect is to select only those entries that match the marked field values. The same conventions apply in this case as for **fill** mode.

7.5 Data Types

Most traditional query languages handle only formatted data in a reasonable manner. However, there are other types of data that may need to be dealt with when querying external data bases. Some of these are: formatted text data nonformatted text data image data, and voice data.

Because of the nature of the different types of data, the way in which they are queried is specific to the data type. However, the same interaction protocols can be used on all types of data.

FORMATTED TEXT DATA

Once we reach the data part (instance level) in the querying process, we are presented with a template of the format of the data part. An example of such a template is shown in figure 6.

We can select from such a template by using the **fill** and/or **display** specification modes discussed above. The default for field filling assumes that we want an exact match for the value specified. For certain types of fields we can also allow other kinds of matches such as pattern matches (find a certain pattern in a field), less than, less than or equal, greater than, greater than or equal and not equal matches. These options can be indicated by prefixing the specified value with a suitable symbol, e.g., $<$, $\leqslant$, etc. Fields to be filled can be selected using the pointing operation discussed earlier.

NONFORMATTED TEXT DATA

If the data part consists of nonformatted text data, then when we arrive at the data part we are again presented with a template of the nonformatted text data. An example of such a template is shown in figure 7.

We can select from text by specifying a pattern for which we are looking in the text. We do this by filling in the text pattern in the appropriate field. Displaying the specified pattern causes the system to

**** VIDEOTEX INFORMATION SERVICE ****

ROUTE: Books.Novel.Gone with the Wind.Chapter 1

MESSAGE ____________________

Fig. 7 Example nonformatted data.

show us the pattern in context, i.e., where it appears in the text. In this case, the *next* and *previous* options for the **entry** operation will show us the next and previous entries with the pattern in context (if the current field is the template). We can also look at other frames around the current frame with the **frame** operation. Entering **display** mode without form filling will show us all the text.

IMAGE AND VOICE DATA

Certain fields within formatted or nonformatted text data may correspond to image or voice data. Currently it is very difficult to query these types of data directly. In the future it may be possible to input a picture or a voice message and ask the system to find all instances of these. However, for now it is only possible to display or play back the image or voice data. Accordingly, this playback or display action can be initiated by selecting the appropriate entry within an image or voice field by the **select** function. In order to indicate to the user the nature of these fields, they can be suitably highlighted by specific colours or marked by special graphic symbols, e.g., an icon of a camera to indicate photographic pictures.

8 CONCLUDING REMARKS

In this report, we examined the process of querying via an interactive query language in general and with respect to external data bases in particular. We first divided the querying process into three parts: request, reply and dynamics. For each part we identified several characteristics and indicated desirable properties of these characteristics. We then identified seven categories of query languages and evaluated each type with respect to these characteristics. From this evaluation, we determined some desirable properties for a query language for external data bases. Finally, we proposed a design for such a query language.

In developing our design for an interactive query language for external data bases we concentrated on the request and dynamics aspects of the query language. For requests the parameters of user interaction were: number of keystokes required, type and number of commands available, degree of freedom in request formulation, selectivity of requests, uniformity of requests with respect to data bases accessed and the ability to customize the request formulation for a specific application. For the dynamics of interaction the parameters related to: the bandwidth between the user and the system, the degree of gamesmanship involved in the interaction, number of protocols available for interaction, the responsiveness of the system to a user request and the degree of control experienced by the user in an interaction. Let us examine the proposed design with respect to these properties.

The number of keystrokes required to specify a request depends on the specification mode chosen by the user. If the user displays and marks entries, then he need only point at, mark and select values. The number of keystrokes required will depend on the pointing mechanism available. Marking requires one keystroke for each value marked while selecting also requires one keystroke for each value selected. If fill mode is used, then the user may have to point at fields and type in field values. However, even in this case many keystrokes can be avoided if the user displays field values and marks them.

In the proposed design there are two specification modes, eight control functions and possibly (depending on the pointing mechanism) one to three pointing operations. One of the two specification modes will be the default mode so the user need not specify it explicitly. Of the eight control functions, two are likely to be used heavily — **select** and **mark**. If these functions are implemented as function buttons and labelled with the function names, then there will be no need for the user to remember the function names. In addition, the functions are named so as to clearly convey their meaning and on-line help is always available via the **help** function.

Because of the way the user interaction has been designed, it is very difficult to formulate a request incorrectly, either syntactically or semantically. Syntactically, the query language has no syntax to remember. The specification modes accept any input state for the templates, from empty to fully filled. Control functions are always valid although they may return a null result. Semantically, field names always appear in the portal and the user can see the structure of the data via the template. It is possible to specify a complicated request incorrectly, e.g., if there are many fields to be qualified with *and*'s and *or*'s. However, the way in which fields on a template interact by default (*and*'s between fields, *or*'s within a field) is fairly intuitive. It must be accepted that complicated requests *are complex* and therefore prone to error in their specification.

The selectivity of the query language is almost as good as keyword and by example query languages. The only difficult operation to specify is a join-like operation (in fact we have not said how to do it). One possibility is to use a saved interaction as input to another interaction. However, we do not feel that there will be a general need for a join-like capability in a query language for external data bases. In any case, one can always have the option of going to a more experienced user interface, i.e., a keyword query language, for such requests.

In the proposed query language, we did not make any assumptions about the structure of the external data bases. The nature of the interface chosen — templates — does not presuppose any particular underlying structure for the data base. Thus, the query language can be used to interface to any type of data base provided suitable translation algorithms are provided to translate the operations and structures in the query language to the operations and structures of the DBMS of the external data base. This is currently an active research area and several approaches to the problem can be taken [Date, 1980; Vassiliou and Lochovsky, 1980].

The query language can be easily customized to particular applications by customizing the templates via which the user interacts with the system. No other aspect of the query language need change to accommodate this customizing. In particular, the hardware required for interaction and the pointing operations, control functions and specification modes remain the same.

The bandwidth of interaction achieved depends on the mode of specification chosen. For display and marking it can be very high. For template filling it can be moderate. Note that the basic concepts behind the interaction remain unchanged as we move from pointing via a keyboard to a mechanical device to voice. Thus, as technology advances allow the bandwidth to increase, the query language can take advantage of these advances.

The gamesmanship of the query language is moderate to good. The user can easily explore alternate paths in the structure with little effort since he can easily save paths and/or back up at any time. This exploration process gradually allows the user to move toward his goal and he can see his progress by the **ROUTE** field in the portal. Depending on how the cross references are constructed, the user can reach the same data from many different paths. The pointing, marking, displaying and filling types of activities involve the user in the interaction. It is not just a matter of pushing a button. The design of the templates and the use of colour and sound can add to the gamesmanship of the query language.

There are several protocols for querying data bases — fill, display and predefined routes. The user can choose the one he feels is most effective for his particular interaction and skill level. He can also set the default specification mode to suit his knowledge level of the system.

Responsiveness may be a limiting factor for the query language with today's technology. **Display** mode especially requires that the system quickly display the requested information. When we are at the data part level this can mean the retrieval of a great deal of information. Being able to display the values of any field implies that there is some fast access mechanism available to quickly retrieve all the values. The use of **display** mode may have to be limited initially to the structure part and certain fields of the data part to overcome the response problems. However, advances in technology may soon make such a restriction unnecessary.

Finally, the entire nature of the user interaction, directing the system to display things, mark things, etc., contributes to the user's feeling of control. He is directing the system to do things for him in reaching his goal. Also, the paradigm for querying introduced earlier helps to contribute to this feeling of control. The paradigm makes the user feel that the system is his friend helping him guide his space ship (the system) through the unfriendly environment of space (the information space) and protecting him from it.

REFERENCES

Bonczek, R.H., Cash, J., and Whinston, A. [1977]. "A Transformational Grammar-based Query Processor for Access Control in a Planning System," *ACM TODS* **2**, pp. 326-338.

Bonczek, R.H., and Whinston, A. [1977]. "A Generalized Mapping Language for Network Data Structures," *Information Systems* **2**, pp. 171-185.

Bown, H.G., O'Brien, C.D., Sawchuk, W., and Storey, J.R. [1978]. *A General Description of Telidon:- A Canadian Proposal for Videotex Systems*, Tech. note 697-E, Communications Research Centre, Department of Communications, Ottawa.

Bown, H., O'Brien, C.D., Sawchuk, W., Storey, J.R., and Treurniet, W.C. [1979]. "Telidon Videotex and User-related Issues," *Proc. Conf. on Visible Languages.*

Burger, J.F. [1977]. "Data Base Semantics in the EUFID System," *Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks*, pp. 202-214.

Carlson, C.R., and Kaplan, R.S. [1976]. "A Generalized Access Path Model and its Application to Relational Data Base Systems," *Proc. ACM SIGMOD*, pp. 143-154.

Chamberlin, D.D., *et al.* [1976]. "SEQUEL2: A Unified Approach to Data Definition, Manipulation and Control," *IBM Journal of Research and Development* **20**, pp. 560-575.

Codd, E.F., Arnold, R.S., Cadiou, J-M., Chang, C.L., and Roussopoulos, N. [1978]. *RENDEZVOUS Version 1: An Experimental English-language Query Formulation System for Casual Users of Relational Data Bases*, Tech. rep. RJ2144, IBM Research Lab., San Jose, Calif.

Date, C.J. [1976]. "An Architecture for High-level Language Database Extensions," *Proc. ACM SIGMOD*, pp. 101-122.

Date, C.J. [1980]. "An Introduction to the Unified Database Language," *Proc. ACM Intl. Conf. Very Large Data Bases*, pp. 15-32.

Deheneffe, C., and Hennebert, H. [1976]. "NUL: A Navigational User's Language for a Network Structured Data Base," *Proc. ACM SIGMOD*, pp. 135-142.

Denny, G.H. [1977]. *An Introduction to SQL, A Structured Query Language*, Tech. Rep. RA93, IBM Research Lab., San Jose, Calif.

Ellis, C.A., and Nutt, G.J. [1980]. "Office Information Systems and Computer Science," *ACM Computing Surveys* **12**, pp. 27-60.

Gilb, T. and Weinberg, G.M. [1977]. *Humanized Input: Techniques for Reliable Keyed Input.* Winthrop Publishers.

Geudj, R.A., tenHagen, P.J.W., Hopgood, F.R.A., Tucker, H.A., and Duce, D.A. (eds) [1980]. *Methodology of Interaction*, IFIP Workshop on Methodology of Interaction, Seillac, France. North-Holland, Amsterdam.

Hammer, M., Howe, W.G., Kruskal, V.J., and Wladawsky, I. [1977]. "A Very High Level Programming Language for Data Processing Applications," *CACM* **20**, pp. 832-840.

Harris, L.R. [1977] "User Oriented Data Base Query with the ROBOT Natural Language Query System," *Proc. ACM Intl. Conf. Very Large Data Bases*, pp. 303-311.

Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., and Slocum, J. [1978]. "Developing a Natural Language Interface to Complex Data," *ACM TODS* **3**, pp. 105-147.

Herot, C.F. [1980]. "Spatial Management of Data," *ACM TODS* **5**, pp. 493-513.

Hogg, J. [1981]. *TLA: A System for Automating Form Procedures*, M.Sc. thesis, Dept. of Computer Science, Univ. of Toronto.

Hogg, J., Nierstrasz, O.M., and Tsichritzis, D.C. [1981]. "Form Procedures," *in Omega Alpha*, Tsichritzis, D.C. (ed.), Tech. rep. CSRG-127, Computer Systems Research Group, Univ. of Toronto.

Kerschberg, L., Ozkarahan, E.A., and Pacheco, J.E.S. [1976]. "A Synthetic English Query Language for a Relational Associative Processor," *Proc. 2nd Intl. Conf. on Software Engineering*, pp. 505-519.

Lacroix, M., and Pirotte, A. [1977a]. ILL: An English Structured Query Language for Relational Data Bases," *in Architecture and Models in Data Base Management Systems*, (Nijssen, G.M., ed.), pp. 237-260, North-Holland, Amsterdam.

Lacroix, M., and Pirotte, A. [1977b]. "Domain-oriented Relational Languages," *Proc. Intl. Conf. Very Large Data Bases*, pp. 370-378.

Lipson, W., and Lapczak, O. [1976]. *LSL User's Manual*, Tech. note 9, Computer Systems Research Group, Univ. of Toronto.

Luo, D., and Yao, S.B. [1981]. "Form Operations By Example — A Language for Office Information Processing," *Proc. ACM SIGMOD.*

McDonald, N., and Stonebraker, M.R. [1975]. "CUPID - The Friendly Query Language," *Proc. ACM Pacific 75 Regional Conf.*, pp. 127-131.

Mantei, M. [1981]. *Disorientation Behaviour in Person-Computer Interaction*, Ph.D. thesis, Dept. of Communications, Univ. of Southern California (to appear).

Martin, J. [1973]. *Design of Man-Computer Dialogues.* Prentice-Hall, Inc., Englewood Cliffs, N.J.

Mehlmann, M. [1981]. *When People Use Computers — An Approach to Developing an Interface.* Prentice-Hall, Inc., Englewood Cliffs, N.J.

Moorhead, W.G. [1976]. *GXRAM - A Relational Data Base Interface for Graphics*, Tech. rep. RJ1735, IBM Research Lab., San Jose, Calif.

Mylopoulos, J., Borgida, A., Cohen, P., Roussopoulos, N., Tsotsos, J., and Wong, H. [1975]. "TORUS - Natural Language Understanding System for Data Management," *Proc. 4th Intl. Joint Conf. Artifical Intelligence*, pp. 414-421.

Mylopoulos, J., Borgida, A., Cohen, P., Roussopoulos, N., Tsotsos, J., and Wong, H. [1976]. "TORUS: A Step Toward Bridging the Gap Between Data Bases and the Casual User," *Information Systems* **2**, pp. 49-64.

Nierstrasz, O.M. [1981]. *Automatic Coordination and Processing of Electronic Forms in TLA (An Intelligent Office Information System)*, M.Sc. thesis, Dept. of Computer Science, Univ. of Toronto.

Prenner, C.J., and Rowe, L.A. [1977]. *A Data Base Oriented Programming Language*, ERL Memo, Electronic Research Laboratory, Univ. Calif. Berkeley.

Reisner, P. [1977]. "Use of Psychological Experimentation as an Aid to Development of a Query Language," *IEEE Trans. on Software Engineering* **3**, pp. 218-229.

Reisner, P. [1981]. "Human Factors Studies of Database Query Languages: A Survey and Assessment," *ACM Computing Surveys* **13**.

Reisner, P., Boyce, R.F., and Chamberlin, D.D. [1975]. "Human Factors Evaluation of Two Data Base Query Languages - SQUARE and SEQUEL," *Proc. AFIPS* **44**, *NCC*, pp. 447-452.

Sagalowicz, D. [1977]. "IDA: An Intelligent Data Access Program," *Proc. ACM Intl. Conf. Very Large Data Bases*, pp. 293-302.

Schmidt, J.W. [1977]. "Some High Level Language Constructs for Data of Type Relation," *ACM TODS* **2**, pp. 247-261.

Sharman, G.C.H. [1977]. "Update-by-dialogue: An Interactive Approach to Data Base Manipulation," *Proc. ACM SIGMOD*, pp. 21-29.

Shneiderman, B. [1978]. "Improving the Human Factors Aspect of Database Interactions," *ACM TODS* **3**, pp. 417-439.

Shneiderman, B. [1980]. *Software Psychology: Human Factors in Computer and Information Systems.* Prentice-Hall, Inc., Englewood Cliffs, N.J.

Spath, C.R. and Schneider, L.S. [1977]. "A Generalized End-user Facility Architecture for Relational Data Base Systems," *Proc. ACM Intl. Conf. Very Large Data Bases*, pp. 359-369.

Stonebraker, M.R., and Rowe, L.A. [1977]. "Observations on Data Manipulation Languages and Their Embedding in General Purpose Programming Languages," *Proc. ACM Intl. Conf. Very Large Data Bases*, pp. 128-143.

Thomas, J.C. [1977]. "Psychological Issues In Data Base Management," *Proc. ACM Intl. Conf. Very Large Data Bases*, pp. 169-185.

Thomas, J.C., and Gould, J.D. [1975]. "A Psychological Study of Query-by-example," *Proc. AFIPS* **44**, *NCC*, pp. 439-445.

Vandijck, E. [1977]. "Towards a More Familiar Relational Retrieval Language," *Information Systems* **2**, pp. 159-169.

Vassiliou, Y., and Lochovsky, F.H. [1980]. "DBMS Transaction Translation," *Proc. COMPSAC '80*, pp. 89-96.

Woods, W.A. [1973]. "Progress in Natural Language Understanding - An Application to Lunar Geology," *Proc. AFIPS* **42**, *NCC*, pp. 441-450.

Zloof, M.M. [1975a]. "Query-by-example: The Invocation and Definition of Tables and Forms," *Proc. ACM Intl. Conf. Very Large Data Bases*, pp. 1-24.

Zloof, M.M. [1975b]. "Query-by-example," *Proc. AFIPS* **44**, *NCC*, pp. 431-438.

Zloof, M.M. [1977]. "Query-by-example: A Data Base Language," *IBM Systems Journal* **16**, pp. 324-343.

Zloof, M.M. [1980]. *A Language for Office and Business Automation*, IBM Tech. rep. RC8091, Yorktown Heights, New York.

Canadä

Government of Canada
Department of Communications

Gouvernement du Canada
Ministère des Communications