An Interactive Query Language for External Data Bases

F.H. Lochovsky
D.C. Tsichritzis

Computer Systems Research Group
University of Toronto

ABSTRACT

The amount of information that is available in computerized form is growing steadily. Two developments in recent years — 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 paper 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 briefly 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. We then propose some requirements that a query language for external data bases should have. With these requirements in mind we detail a design for such a query language.

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 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 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 are 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 [McDonald and Stonebraker, 1975; Denny, 1977; Zloof, 1977; Codd et al., 1978; Ellis and Nutt, 1980; Herot, 1980]. 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 query language for accessing external data bases via videotex systems that is easy to learn and use by the public. In this paper we outline a framework for evaluating interactive query languages and propose a particular approach to a query language for external data bases.

2 USER INTERACTION PARAMETERS

The general area 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]. As a way of interpreting the results of some of this research and of providing more structure to the human factors evaluation of interactive query languages, we propose that the querying process be viewed as consisting of three parts: request, reply and dynamics. 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 the person-computer interface. In the rest of this section we will identify and define several characteristics for each part of the querying process. For a more detailed discussion of these parameters see [Lochovsky and Tsichritzis, 1981].

2.1 Request

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.). In our opinion, six characteristics of formulating requests are important for evaluating interactive query languages.

1. *Keystrokes* are a quantification of the amount of user-supplied input required before the system fully understands the user's request.
2. *Commands* are the set of instructions a user has available to tell the system what action it should perform.
3. *Formulation* of a request concerns how difficult it is to specify a request that is incorrect either syntactically or semantically.
4. *Selectivity* is the ability of the user to specify as precisely as possible that data which he wishes to retrieve.
5. *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.
6. *Customizing* of a query language deals with matching the form of a query language to that of a particular application.

2.2 Reply

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 with possibly sound also available. In our opinion, five characteristics for representing replies are important for evaluating interactive query languages.

1. *Presentation complexity* is a measure of how long it takes the user to grasp the content of a reply.
2. *Multi-media* is a measure of the number of ways in which the system allows people to communicate.
3. *Customizing* of replies, as for customizing of requests, means that the form of the reply is determined by the environmental factors of the interaction.
4. *Dynamic control* is a measure of the ability of the user to control the pace at which and form in which the reply to a request is presented to him.
5. *Reusability* of a reply is a measure of whether the reply to a request is only for immediate consumption, or may be useful at a later time as input in the formulation of another request.

2.3 Dynamics

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 this interaction to be as easy and interesting for the user as possible. In our opinion, five characteristics of the interaction between the user and the system are important for evaluating interactive query languages.

1. The *bandwidth* of interaction is a measure of the speed at which the user and the system communicate with each other.
2. *Gamesmanship* is a measure of how challenging and interesting the interaction between the user and the system is.
3. *Protocol complexity* is a measure of the number of protocols, and their difficulty, available to the user.
4. *Responsiveness* is a measure of how fast the system responds to a user's request.
5. *Control* is a measure of how comfortable people feel in using a system.

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.

3 QUERYING REQUIREMENTS

In the 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 paper we will concentrate mainly on the request aspects of a query language and its dynamics. In this section, we will indicate what, in our opinion and that of some of the studies cited earlier, constitutes a "desirable" value for some of the characteristics of the request and dynamics parts of the querying process.

In general, a user wishing to access an external data base 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, 1982]. 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. 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.

The user interface should use to advantage the ability of people to remember and easily recognize abstract patterns. For example, people are much better at processing image data than are computers. In addition, they readily associate abstract images, such as logos, with the objects which the images identify. One should make use of the image processing and associative capability of people in a query language.

Finally, the available technology (graphics, colour and sound) should be used to distinguish between different types of things on the screen. For example, we could use different colours to 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 paper, 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.

4 PARADIGM

The paradigm that we choose for accessing information via an interactive query language for external data bases 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.

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.

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 paper we only discuss the "automatic" query language.

5 DESIGN OVERVIEW

In this section we describe the overall design of the query language. We discuss *what* can be done and not *how* to do it. Thus we describe the operations generically and use generic

**** VIDEOTEX INFORMATION SERVICE ****

ROUTE:_____

CATEGORY	CONTENTS

MESSAGE:_____

Figure 1 Empty display template.

terms such as "point at", "fill in" and "select" without discussing how these operations are done. We use the analogy developed in the previous section to explain the operations in the query language. In the next section, we consider how these operations are done by the user.

5.1 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 view screen (TV screen) 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 view screen such as that shown in figure 1.

The exact format of the template in the view screen 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 parts. One part specifies the *path* we have taken to arrive at the current template in the view screen. 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

**** 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

Figure 2 Nonselective display of all available information sources.

**** 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

Figure 3 Nonselective display of Entertainment information source.

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 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

**** VIDEOTEX INFORMATION SERVICE ****

ROUTE: _____

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

MESSAGE: Frame 1 of 1

Figure 4 Selective display of available information sources.

**** VIDEOTEX INFORMATION SERVICE ****

ROUTE: Entertainment.Films.First run

FILM	STARRING	THEATRE	TIMES	PHONE#

MESSAGE:

Figure 5 Empty display template for information source Entertainment.Films.First run.

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 words 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 view screen is a function of how the cross references to data have been designed within the system.

Associated with each keyword displayed in the view screen 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.

5.2 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 the structure part, we can now explore and navigate the data part. As for the structure part, we again are presented with an abstract pattern (template) of the data in the view screen. An example of such a template is shown in figure 5.

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.

5.3 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.

6 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 are done. In this section we elaborate on how the actions discussed in the previous section are performed by a videotex user.

6.1 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 view screen 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 view screen. The pointing mechanism employed in our system is a puck and tablet. In this way, the user is able to position a cursor to the appropriate position within the view screen and to indicate the selection of this position.

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 are independent of each other. The exact semantics of the specification depend on the characteristics of the template displayed in the view screen. The ability to select a specific field and a specific entry leads to the notion of a *current* field and entry. Visually this situation is indicated by highlighting the current entry and field using a different colour.

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. If the current field is the template, then an entry refers to all the values that define one instance of the template.

6.2 Positioning

In our discussion we have not considered any restrictions on the size of the template that could be displayed in the view screen. We implicitly assumed that everything could be displayed at once. In general, however, templates may be larger than the view screen size. Ideally, we would like to be able to move the view screen 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 view screen and continuous motion in all directions is not possible. Accordingly, we 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 view screen 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). 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 view screen. Thus, for example, tables of data that are too wide to fit within the view screen are 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 view screen are divided up into several horizontally adjacent frames (above and/or below each other).

To position the view screen to the desired frame, there is a **frame** operation with the following options and semantics:

1. **frame** *up* — moves the view screen to the frame above the current frame
2. **frame** *down* — moves the view screen to the frame below the current frame
3. **frame** *left* — moves the view screen to the left adjacent frame
4. **frame** *right* — moves the view screen to the right adjacent frame
5. **frame** *name* — moves the view screen to the named frame.

Frames can be named symbolically so that it is possible to position the view screen on them directly. In most cases it is desirable not to have right and left frames as this will fragment the data a great deal.

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. In our system, we 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 view screen. In the vertical direction this is not a great problem since as we scan up and down the same template or the same part of the template is displayed in the view screen. However, in the horizontal direction the part of the template visible in the view screen changes as we move left and right. It is therefore desirable to keep some common thread between horizontally adjacent frames. In our system we designate one of the template fields as a *key* field and keep this field in the view screen 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 view screen as we move the view screen position horizontally.

6.3 Control Functions

Control functions deal with signalling the system that it should note something about its current state 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 are provided as function buttons in our system.

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.

UNDO

The **undo** 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 plus optionally a user supplied name. 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 a local 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. Saved interactions are kept in a local data base named 'saved'. 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.

DROP

The **drop** function takes as its input parameter the current field value. If the current field value is a saved interaction, then the result of the function is to drop the saved interaction from the structure part information for this user. Dropping a non-saved interaction results in a null action.

6.4 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 **undo** 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**, **suspend**, **save** or **drop** function.

FILL

Fill mode allows the user to select various fields and to specify a value or values for each field selected [Zloof, 1975a,b, 1977]. 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 is 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 view screen. 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 view screen 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 selects 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.

6.5 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

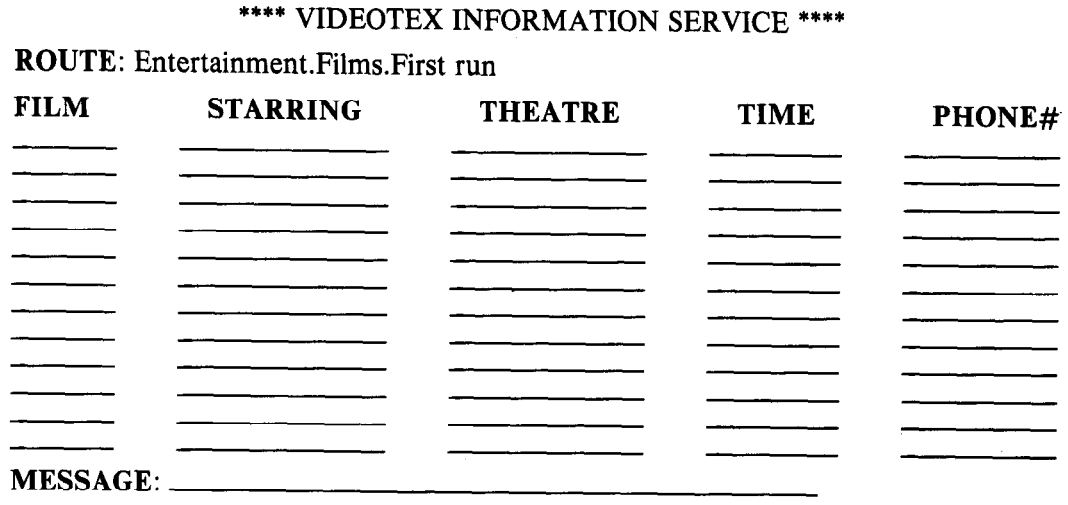

Figure 6 Display template for formatted data.

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 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 are indicated by prefixing the specified value with a suitable symbol (e.g., $<$, $\leqslant$, etc.) [Zloof, 1977]. Fields to be filled are 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 select from text by specifying a pattern for which we are looking in the text [Tsichritzis and Christodoulakis, 1982]. We do this by filling in the text pattern in the appropriate field. Displaying the specified pattern causes the system to 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 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 shows 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 them. However, for now it is only possible to display or play back the image or voice data. Accordingly, this playback or display action is 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 are marked by special graphic symbols (e.g., an icon of a camera to indicate photographic pictures).

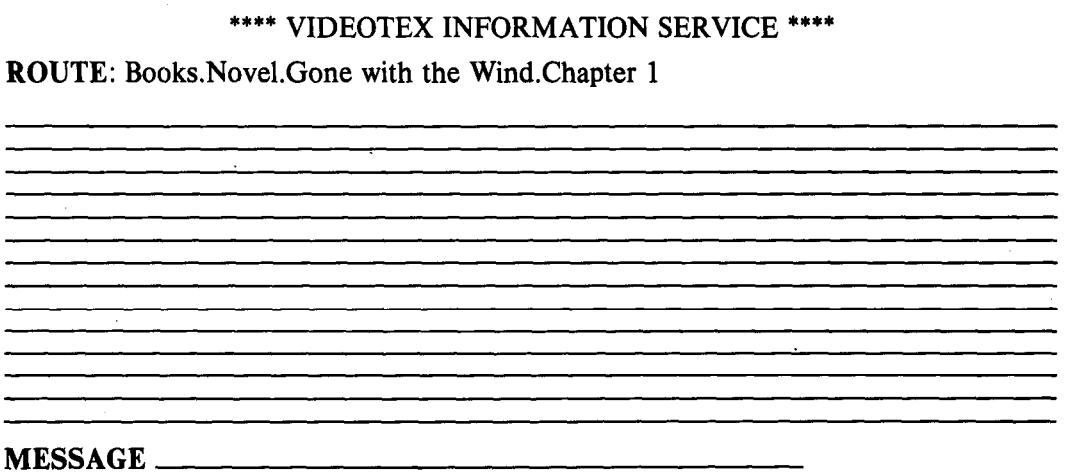

Figure 7 Display template for nonformatted data.

7 DISCUSSION OF THE QUERY LANGUAGE DESIGN

In this paper, we examined the process of querying via an interactive query language in general and with respect to external data bases in particular. In developing our design, we concentrated on the request and dynamics aspects of the query language. Let us examine the proposed design with respect to the characteristics of these parameters.

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. 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, nine control functions, one pointing operation and one positioning operation. One of the two specification modes will be the default mode so the user need not specify it explicitly. Of the nine control functions, two are likely to be used heavily — **select** and **mark**. Since the control functions are implemented as function buttons and labelled with the function names, there is 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 view screen 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. We feel that 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 through 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 and positioning 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 mechanical device to, say, voice-directed pointing. 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 view screen. 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. 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.

8 SUMMARY

The advent of videotex systems and the growth in communications networks are making feasible access to a variety of computerized information sources by people not familiar with computer technology. In order to make effective use of these information sources, access to external data bases will have to be via an easy to use interactive query language. In this paper, we proposed a design for such a query language.

We first considered the process of accessing a data base (querying) in terms of the user interaction characteristics of the process. We divided the querying process into three parts: request, reply and dynamics. For all three parts, certain characteristics were 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 some requirements that a query language for external data bases should have in terms of a desirable "value" for the parameter were identified. With these requirements in mind we then proposed 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 was 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.

The query language is being implemented in C running under UNIX on a VAX-11/780. Initially, relational data bases will be accessed, but the capability to access DBTG-network data bases will also be investigated.

ACKNOWLEDGEMENT

This research was sponsored by the Department of Communications, Ottawa, Canada under Department of Supply and Services contract serial no. 0SU80-00124.

REFERENCES

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.*

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

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.

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

Lochovsky, F.H., and Tsichritzis, D.C. [1981]. *Interactive Query Languages for External Data Bases.* Dept. of Communications, Ottawa, Canada.

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

Mantei, M. [1982]. *Disorientation Behaviour in Person-Computer Interaction*, Ph.D. thesis, Annenberg School of Communications, Univ. of Southern California.

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.

Reisner, P. [1981]. "Human Factors Studies of Database Query Languages: A Survey and Assessment," *ACM Computing Surveys* **13**, pp. 13-31.

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

Tsichritzis, D.C., and Christodoulakis, S. [1982]. "Message Files," *Proc. ACM SIGOA Conf. on Office Information Systems*, pp. 110-112.

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

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.