UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT

Graphical Specification of Keyword Queries to Videotex Databases

Vilhelm Böggild
Data Structuring Group

CS-86-63

November 1986

Graphical Specification of Keyword Queries to Videotex Databases†

Vilhelm Böggild

Department of Computer Science
University of Waterloo
October 19, 1986

ABSTRACT

Information systems must continue to improve their query facilities as the needs and expectations of our information-based society grow. This need is perhaps felt most acutely in videotex and other public database systems. However, such systems have in general not exploited their graphics capabilities to aid in information retrieval.

A recognition experiment was conducted to test the understanding of Venn diagrams. The results indicate that people's understanding of Venn diagrams is more complete and complex than had been previously recognized. In particular, the use of negation in descriptions was surprisingly prominent.

A prototype keyword system was developed for a videotex database that allows for the graphical specification of queries via Venn diagrams. The system was developed in a distributed, multi-process environment. Various aspects of process structuring and features of the hardware configuration are examined.

As a result of these two activities, it can be concluded that systems employing a graphical keyword interface for page retrieval should be further investigated.

October 19, 1986

---

† Originally a master's thesis presented to the Faculty of Mathematics at the University of Waterloo.

Acknowledgements

Produced by ☞ *Frank Wm. Tompa*

I would like to thank Frank Wm. Tompa for being my supervisor during my stay at the University of Waterloo. He knew when to let me investigate areas on my own and when to focus my attention on a certain area (like getting the thesis done).

Financed by ☞ *NSERC, U of W*

Financial support came from various sources throughout the course of this work and was very much appreciated: NSERC strategic grant G1154, NSERC post graduate scholarship, and the University of Waterloo in the form of teaching assistantships.

Directed by ☞ *Darrell Raymond*

I would like to thank Darrell Raymond for his continued support and suggestions throughout the development of this thesis. Life would have been quite a bit rougher than normal without his help.

Edited by ☞ *Doug Dyment* and *Frank Safayeni*

I would like to thank my readers for their helpful suggestions and comments.

The following people have also contributed in ways too numerous to mention.

Supporting cast

Steve Williams	R.A. student
Paul Böggild	Father
Faye Böggild	Mother
Paula Böggild	Sister
Michelle Dunkley	girlfriend
Greg Johnston	roommate
Waterloo Warriors Basketball Team	themselves
Lou	chef at Renison College
Elsa	dog

Table of Contents

Introduction

What is videotex? Videotex was originally viewed as a simple information retrieval system for the home, using an adapted TV set with a key pad. The kinds of information a person might retrieve using a videotex system vary from news, weather, and entertainment, to interactions with banks and stores. Now public terminals also exist in places such as shopping malls and bus stations to provide information about attractions, dining, maps, transportation schedules, and so on. This simplistic view is changing because customers expect better information retrieval and access methods, and systems are evolving by the introduction of more complex features.

There are three features common to most videotex systems. First, there is an extensive use of computer generated images or graphics. Second, the information is displayed in a page format, one screen at a time. Third, the pages of information in the videotex database are structured in a hierarchy and are typically accessed by a set of hierarchical menus.

Menu-based access to information means that users who know little about computers or the available database can still find information. However, menu hierarchies can be confusing. People view the world in different ways and so seldom agree universally how information should be categorized. Miscategorization of information is a most serious design defect, leading to errors or an inability to find information at all [DOC 81a]. Several alternative access methods exist, such as labels and lists, keywords, cross-links, multiple contexts, lattices, and multi-menus [Raymond 84].

The need for simple and powerful access methods and query languages for public information systems such as videotex is clear. However, the proper design of a suitable method and language to meet these criteria is not as clear. The main contribution of this thesis is to highlight one approach to reduce retrieval disadvantages of videotex by exploiting its graphical advantages:

> "Why is it that simple graphic schemes (Venn diagrams, boxes, arrows, etc.) are so popular in scientific and educational contexts? Their popularity, it is proposed, is linked to their ability to help us grasp generic properties and underlying relations not directly observable in the surface features of physical phenomena." [Mills 81]

A graphical query language has been developed for a relational database that employs Venn diagrams in the construction of boolean queries [Michard 82]. Michard's use of Venn diagrams in the query language and his favourable ergonomic evaluation of the system prompted further investigation into the possible use of Venn diagrams for querying videotex databases.

The prototype keyword system utilizes windows, graphics, and a mouse to assist the user in specifying a query. Existing videotex systems do not allow for the combination of keywords using the rules of boolean logic, nor do they use graphics and a pointing device to aid the user.

Section 2 outlines the background material relevant to the development of the keyword system. Michard's GQL is described and his ergonomic evaluation is examined. The history of logic diagrams and some of their interesting characteristics are noted. A recognition experiment was carried out to test if people could use natural language to describe queries that had been given in Venn diagram form. The results from this experiment supported the development of a prototype keyword system employing Venn diagrams for querying a videotex database. The use of keywords in videotex systems is discussed, followed by an overview of the prototype system.

The implementation of the keyword system required the local processing power of a microcomputer in addition to that associated with the typical videotex environment. The hardware configuration and associated issues are discussed in Section 3.

The keyword system was designed and implemented with a multiprocess, message passing operating system and programming language. Various features of this environment are examined including anthropomorphic programming models and a new process manager model. The design and implementation of the prototype system with the process manager model is contained in Section 5.

Section 6 reviews the problems encountered during the development of the system and contains suggestions for further enhancement of the system and concluding remarks.

A Keyword Interface Based on Venn Diagrams

2.1. Michard's Graphical Query Language

Michard discusses a new query language, known as the Graphical Query Language or *GQL*, suitable for naive users of a single relation (one table) database [Michard 82]. The main feature of GQL is the replacement of formal boolean expressions by Venn diagrams.

In the pilot GQL system the user is presented with a screen containing soft buttons and a display area for a Venn diagram, as shown in Diagram 2.1. Among the soft buttons are the attribute names of the single relation which are contained in the long horizontal box at the top of the display area. Four possible attribute value buttons are available including a numeric keypad, relational operators ($>, <, =, \neq, \geq, \leq$) and a command area, containing functions such as *enter*, *cancel*, *memorize*, *search* and *end*.

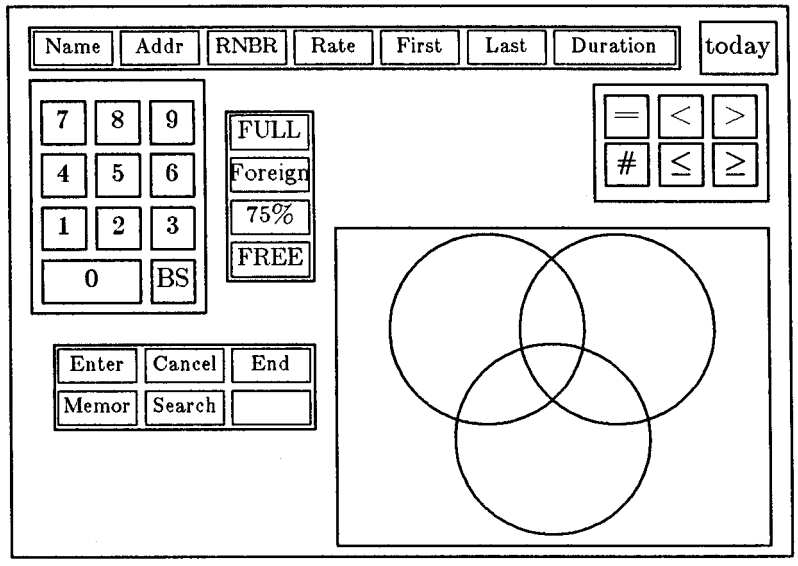


Diagram 2.1

To pose a query, the user formulates a query criterion, such as "RATE = FULL". Each time a criterion is formulated, a numbered circle appears in the Venn diagram area with the corresponding legend indicating the selection to which it corresponds. When another criterion is selected, an additional circle appears overlapping the first, as would be expected in a Venn diagram. By pointing, the user is able to select various regions within the Venn diagram and these areas become hatched. The user can then select the search command to invoke the query represented by the hatched region in the Venn diagram.

Diagram 2.2 shows the construction of the query "Select everyone in the database living in region 75 or in regions 78 and over, and who is subscribing at the full rate."

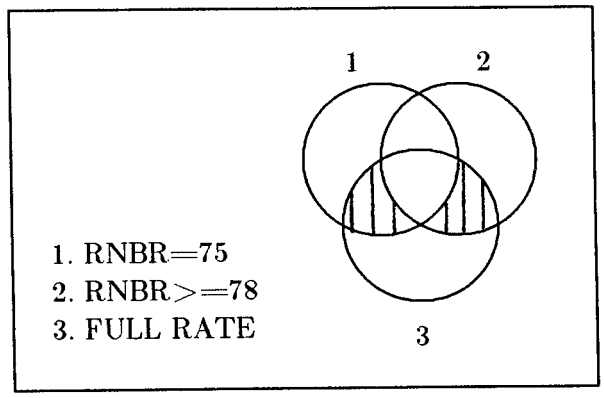


Diagram 2.2

The Venn diagram has been limited to a maximum of three circles in GQL. For complex queries involving more than three criteria, the command MEMOR allows the currently selected subsets to be represented by a single circle in the diagram. Further selection criteria may be chosen, and these can be encapsulated in turn and re-used in a more complex query.

Michard identifies the following important features that are present in the design of GQL:

- *non-procedural* — the user constructs a diagram representing the desired result as opposed to specifying how to find the result.
- *free order* — the user can construct the Venn diagram in different ways with the same items to achieve the same result.
- *immediate feedback* — all actions initiated by the user have an immediate effect on the display.
- *aid in the manipulation of boolean operators for set operations* — Venn diagrams are utilized to "visualize" the contents of selected subsets.
- *menu-oriented dialogue* — all commands are evoked by selecting soft buttons on the display device.
- *external representation of the data structure* — similar to QBE (Query-By-Example), the relational scheme is itself a significant component of the display.

Immediate feedback and menu-oriented dialogue design features are desirable for any system for use by casual users. The other design features are worthwhile, but the most intriguing feature is the use of Venn diagrams to construct queries. While such a method is intuitively appealing, Michard does not address the complex mechanism underlying comprehension of Venn diagrams.

Michard's system lacks a description of how query results should be handled and displayed; there does not appear to be a window for this purpose. A visually-oriented query facility such as this should have a graphical feature for manipulating query results.

The use of Venn diagrams in interactive graphical query systems should be examined more closely to see how they are more expressive or understandable than boolean expressions, and where the power of this alternative form lies.

2.2. Logic diagrams

John Venn was born in Hull, England, on August 4, 1834, and died in Cambridge, England, on April 4, 1923. Venn's volumes on probability and logic were highly esteemed textbooks in the late nineteenth and early twentieth centuries [Dict. Sci. Biography 76]. The use of geometric diagrams to represent syllogistic logic has a long and interesting history and has been succinctly outlined by M.E. Baron, "A Note on the Historical Development of Logic Diagrams: Leibniz, Euler and Venn" [Baron 69]. Syllogism refers to a form of reasoning in which two premises are made and a logical conclusion drawn from them [Webster Dict. 83].

It was Gottfried Wilhelm Leibniz (1646-1716), who first devoted serious study to the analysis of logical propositions by means of diagrams rather than using them as casual illustrations. He explored the possibility of representing syllogistic arguments by means of geometric figures developing the familiar circle diagrams attributed to Venn and Euler, as well as an alternative linear form [Baron 69]. The popularization of circle diagrams was largely due to Euler from a paper he wrote in 1795. In describing the diagrams he thought they would be of great assistance to comprehending and would exhibit the nature of the problems to the eye. The value of having a visual graphical notation was recognized, but not necessarily understood.

Venn examined logic diagrams carefully and surveyed the contributions of his contemporaries. Venn brought about a clarification of symbolic logic and the algebra of classes developed previously in 1847 by George Boole. Venn thought that the Eulerian scheme was weak and developed a more general framework to which the regular system of class subdivision would correspond. For one term, two compartments are needed (x,$\bar{x}$); for two terms, four compartments are needed (xy,$\bar{x}y$,$x\bar{y}$,$\overline{xy}$); and in general for n terms there are 2^n compartments. The convex closed figures in Diagram 2.3 represent one, two, three and four terms.

One should note that the figure with four terms is comprised of ellipses and considerably harder to interpret, not being as regularly symmetric as the previous figure. Such a figure is impossible to draw with four circles where only thirteen of the fifteen regions can be shown.

From Diagram 2.4 it is apparent that the symmetry of the figures does not allow for the two extra regions where only the two diagonally opposite circles intersect. The missing regions are attributable to the fact that circles can intersect at a maximum of two points (they could

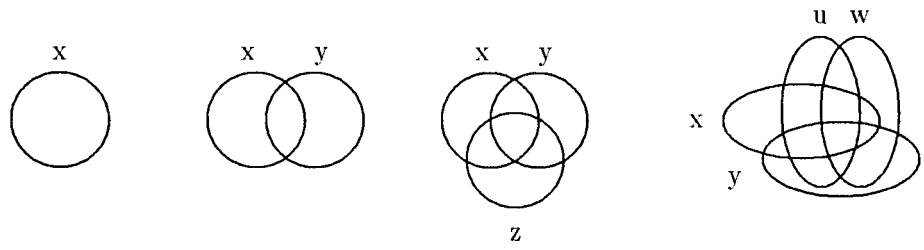


Diagram 2.3

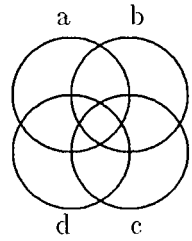


Diagram 2.4

intersect at all points, but then only one of the two circles would be visible).

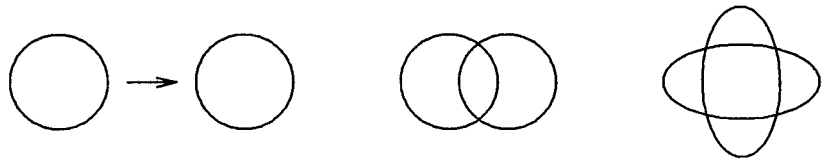


Diagram 2.5

When an additional circle is added to a diagram, the new circle will create at most two new regions with each of the existing circles. Therefore 2(n-1) more regions are created for each additional circle. With four circles, only 13 regions are possible. However, two ellipses can intersect at four points to form five regions, though some of the regions represent the same subset (see Diagram 2.5.c). This feature allows for the construction of the four term figure in Diagram 2.3. Diagrams with more than four terms are very complex and involve non-convex figures, so no attempt will be made to illustrate them here. For a discussion of these and other diagrams, see pages 122-124 of Baron's article [Baron 69].

It was therefore a wise decision to limit GQL's Venn diagram to three circles to balance completeness and comprehensiveness. Michard dismisses instances of greater than three circles within GQL with the casual observation that :

> "The number of simultaneous criteria that can be treated in an elementary query has been limited to three, for pointing on a Venn diagram involving more than three circles would become quite involved."

More importantly, complete diagrams of more than 3 criteria are not possible using circles. The scheme employed by Michard to encapsulate previous selections into a single circle in order to continue with complex queries of greater than three criteria is one means of avoiding the problem.

2.3. Experiments

2.3.1. Michard's evaluation

Michard investigated the virtues of GQL compared to traditional boolean query languages for casual users [Michard 82]. An ad hoc query language was implemented in the same style as GQL except that boolean expressions were constructed with traditional symbols (AND, OR, MINUS and PARENTHESES) and grammar. Twelve college students with no previous computer experience (but a knowledge of basic boolean algebra and Venn diagrams) were used as subjects. All subjects learned both languages to control the transfer of learning; half the subjects learned GQL first while the other half learned the "test" language first. Each subject was given a training session involving a series of 18 query problems of increasing complexity. During the training session, subjects were given extensive help; each error was corrected and explained as it occurred. The experimental task involved translating eight natural language specifications into an appropriate boolean query for each language.

Incorrect set operations were counted as errors. Also, missing brackets in the "test" language expressions were considered as errors even if the correct selection was made from the expression due to the left to right parsing of the implementation. Bracket misuses formed the main difference in error scores between the two languages. As a result, the data show that subjects using the GQL for set operations do much better than those using the "test" boolean query language.

2.3.2. Venn diagrams

Michard's experiment showed that subjects made fewer querying errors using GQL, but he did not explain the power of using a Venn diagram in the GQL interface other than it being an interactive graphic assistance to subset selection. Michard's data support the observation that constructing boolean expressions is difficult [Cooper 82] [Vigil 83]. Although part of the difficulty stems from the complexity of querying, the human factors aspects of the language are also significant. For example, a common misconception is that *(A and B)* contains more than *A*, because "and" can be easily misinterpreted as union. However, correct identification of *(A and B)* in a Venn diagram is supported by the diagram's

graphical message that *(A and B)* is smaller than either *A or B*.

Venn diagrams intuitively seem to be a better way to create a boolean specification for a query. The popularity of simple graphic schemes in science and education is proposed to be linked to their ability to help us grasp generic properties and underlying relations [Mills 81]. In an attempt to explore the nature of Venn diagrams further, another experiment was designed with an application for a keyword querying facility in mind. This experiment differs from Michard's in that subjects were asked to describe in natural language various regions that were marked on Venn diagrams. It was thought that this would eliminate bias resulting from an experimental design requiring the experimenter to describe the query with boolean operators in natural language, as in Michard's experiment. Another benefit from this approach is that many descriptions would be generated that could be given to subjects in a follow-up experiment. It would be interesting to see if other subjects would be able to identify the correct regions from the generated descriptions. In particular, logically incorrect descriptions that are subsequently interpreted correctly could be contrasted with logically correct descriptions that are subsequently falsely interpreted.

2.3.2.1. Description

The experiment involved thirty-two undergraduate students who were given seven Venn diagrams, each with one or more shaded regions. The subjects were asked to describe the shaded regions in such a way that another person could identify the regions in question on a similar diagram which did not have the shaded regions. The subjects were shown a sample diagram and sample solution. See Appendix A.1 for the specific instruction stimuli used in the experiment. The subjects had an unlimited time to perform the task, and were paid a small sum for their participation. All subjects indicated that they had seen Venn diagrams before.

To test diagrams of varying complexity, two of the seven Venn diagrams had two circles and the remaining five contained three circles. Also, the number of shaded regions in the diagrams ranged from one to four. Relational attribute expressions were used as labels in Michard's experiment; in this experiment plausible keywords were used as labels for each circle. Two types of labels were used on the diagrams: labels were either general categories (science, history, and art) or they were specific (music, transportation, and shops). Half of the subjects received general labels on the two-circle diagrams and specific labels on the three-circle diagrams. The other sixteen subjects received specific labels for the two-circle diagrams and general labels on the remaining three-circle diagrams. The data retrieved from the experiment are in Appendix A.2 organized by diagram.

The descriptions for each diagram were judged to be either correct or incorrect according to boolean logic despite having the descriptions in natural language. This stringent scoring scheme was used because it was very difficult to decide for some descriptions which were right or wrong.

Some people used the rules of boolean logic and, as expected, some people used "and" for union within their descriptions instead of boolean "or". Still others avoided the "and/or problem" by using other connectives and phrases. A less stringent scoring scheme could be used to differentiate those descriptions which were reasonable from those which were incongruous, and this would increase the score for the number of correct descriptions. Alternatively, a multi-valued scoring scheme could be used to rank the descriptions.

Diagrams were chosen for perceived difficulty as well as the need to use negation in a description. Wason and Johnson-Laird describe negation as a fundamental cognitive process that is intrinsic and indispensable to reasoning [Wason 72]. However, they state that negation is the most difficult of the three logic operators to comprehend and support this with experimental data using negation in true and false questions. Vigil reports that users of online literary search systems rarely use boolean NOT and that instructional manuals for these systems caution against its use [Vigil 83]. The experiment uses two diagrams representing the queries with the most errors in Michard's experiment. Diagram 2.7 represents the query: *(A or B) and not (A and B).* Diagram 2.8 represents the query: *(A and not B) and C.* Other diagrams represent queries not used by Michard such as coordination-level match queries of the type "find all documents with at least two of these three keywords."

2.3.2.2. Results

Since each diagram had its own potential for difficulty, different tallies were made for each that would not be applicable to the whole set. The following paragraphs discuss each test diagram and the results obtained.

The first diagram is a simple case of one shaded region in a two circle diagram. The corresponding boolean representation for this diagram is : *A and not B.*

Diagram 2.6 Stimuli for condition 1

As expected, no one made an error in either the general or specific label group. Only two of the subjects did not use negation in their description. Of the rest, a wide variety of negation words were employed: *excluding, doesn't include, except, eliminate, everything but, only, other than, "non-"* and *omit.* Such words are used 36% of the time in five of

Results for condition 1			
label type	correct	incorrect	use of negation
general	16	0	14
specific	16	0	16

Table 2.1

the seven test conditions where a description required the use of negation. The remaining 64% comprised descriptions that explicitly used *not* and those that used examples.

The second diagram was more complex than the first, and the subjects made errors half the time. Two possible boolean representations are:

(i) *(A and not B) or (B and not A)* { disjunctive form }
(ii) *(A or B) and not (B and A)* { conjunctive form }

Diagram 2.7 Stimuli for condition 2

Results for condition 2				
label type	correct	incorrect	use of negation	AND/OR problem
general	7	9	14	3
specific	8	8	16	4

Table 2.2

Seven of the seventeen errors made were a direct result of confusion about which of "and" and "or" is union and which is intersection. One fifth of the subjects assumed that the negation of the intersection implied the selection of the two remaining regions. This seems to indicate that people feel comfortable with the notion of relative complement, but do not understand how it differs from universal complement.

The third diagram was the first of the group of five with three circles. A boolean representation is: *not A and B and C*

Descriptions were counted as being wrong if they were not restrictive enough, even if the indicated region was contained in the description. This strict rule affected the specific group where five descriptions could potentially be considered correct. Over half of the subjects had incorrect descriptions in the specific label diagram, but only three had an error when general labels were used. Those subjects who had the general labels

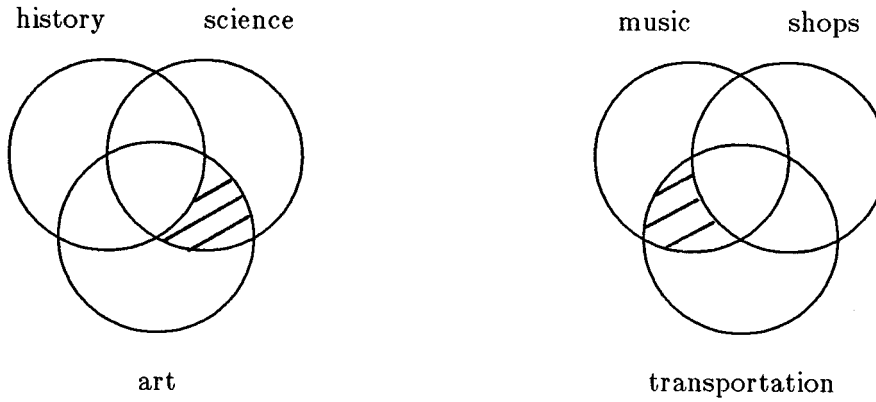


Diagram 2.8 Stimuli for condition 3

Results for condition 3			
label type	correct	incorrect	use of negation
general	13	3	13
specific	6	10	10

Table 2.3

were able to use them as adjectives in their descriptions quite easily. The subjects with the specific labels could not do so as easily and instead attempted to describe the region by use of a sample element. These sample elements tended to be too general and hence could possibly include more than the selected region.

The fourth diagram contained two shaded regions and three criteria, of which only two were needed in the description. The boolean representation for this diagram is: *A and B*

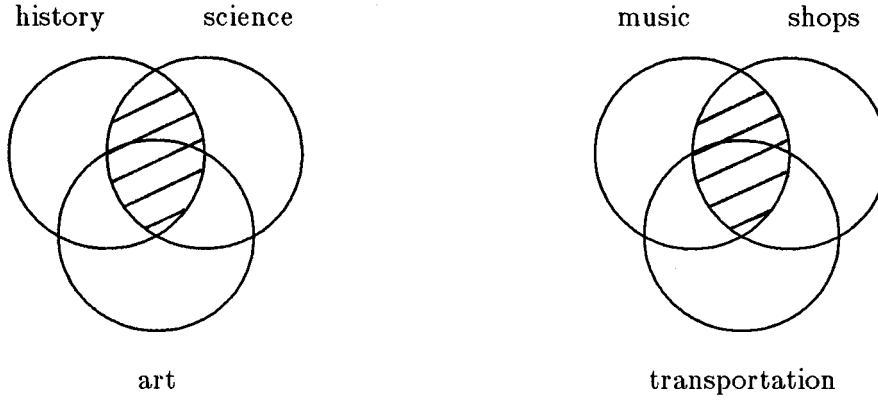


Diagram 2.9 Stimuli for condition 4

Results for condition 4			
label type	correct	incorrect	combination
general	13	3	4
specific	10	6	8

Table 2.4

The most common errors were descriptions that were too restrictive, and attempts to incorporate the third label in the description.

The fifth diagram was again another example of a single region that was marked on the diagram, but it is different from Diagram 2.8 because it represents a query where the two other labels should be excluded from the set. Also note the drop in errors from Diagram 2.8 to 2.10. A boolean expression for this is: *B and not (A or C)* {"and not" is the same as MINUS}.

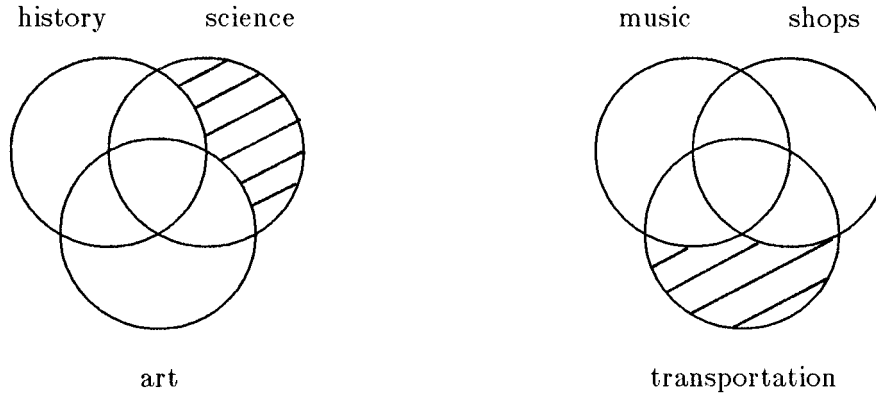


Diagram 2.10 Stimuli for condition 5

Results for condition 5				
label type	correct	incorrect	use of negation	examples
general	16	0	13	2
specific	14	2	6	7

Table 2.5

All sixteen subjects in the general label case had correct descriptions. In the specific case, seven subjects attempted to use examples whereas none did in the general case. As a result of using examples, fewer people in the specific group used negation in their descriptions.

The sixth diagram has four marked regions and is representative of a coordination-level match query where the user is trying to find documents with two or three of the indicated keywords. An equivalent boolean expression is: *(A and B) or (B and C) or (C and A)*

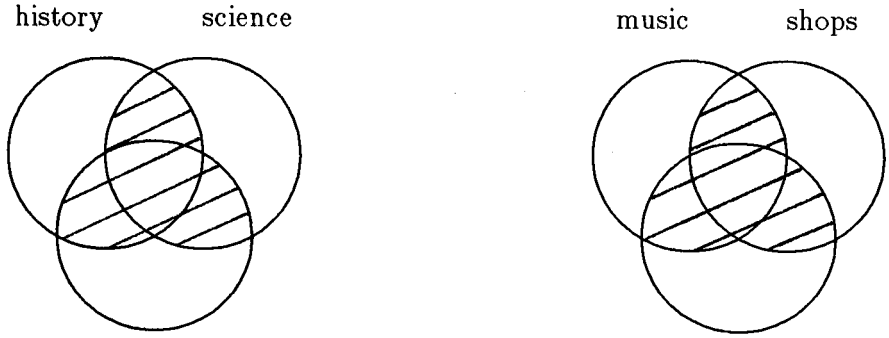


Diagram 2.11 Stimuli for condition 6

Results for condition 6					
label type	correct	incorrect	coordination level query	examples	confusing
general	13	3	7	2	2
specific	8	8	2	7	7

Table 2.6

The general label group performed significantly better than the specific group. Almost half of the descriptions in the general group indicated that the diagram was representative of a coordination-level match query, whereas only two recognized this in the specific label group. Almost half of the specific group still attempted to use examples for their descriptions which resulted in very confusing and rather bizarre descriptions.

The seventh diagram was very similar to the sixth except that it was missing the area of intersection between all three keywords (e.g. a coordination level match for exactly two criteria). Two possible boolean expressions are:

(i) *((A and B) or (B and C) or (C and A)) and not (A and B and C)*
(ii) *(A and B and not C) or (A and not B and C) or (not A and B and C)*

The results are similar to the previous diagram except that some people were unable to give a description. Again, some of the descriptions of the specific group were very confusing.

2.3.2.3. Conclusions

There was a difference in performance between the subjects with general labels and those with specific labels. General labels such as science and history can be combined to form phrases such as scientific history and historical science, which identify the same region of the diagram. However, with labels such as music and shops, one can construct "musical shops" but shop cannot be easily converted to an adjective to modify music. When specific labels were used, more subjects felt it necessary to

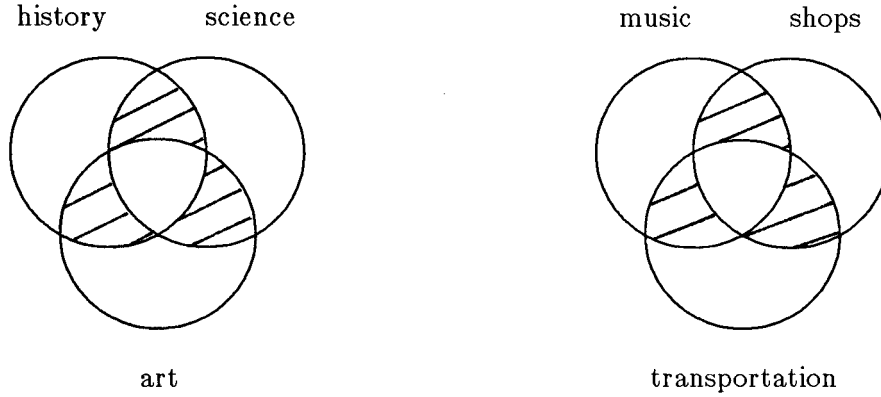


Diagram 2.12 Stimuli for condition 7

Results for condition 7						
label type	correct	wrong	no answer	coord. query	examples	confusing
general	12	3	1	7	1	1
specific	7	7	2	3	6	6

Table 2.7

employ examples to describe regions. This method caused some problems in the more complicated diagrams and as a result, the group with the general labels performed better. The subjects with the specific labels were more restricted in their possible descriptive phrase constructions and were attempting to describe a smaller domain. It seems as if the perceived size or significance of the intersections plays a role in deciding which descriptive method to use. Another factor is the meaningfulness of the categories relative to each other. In any case, the significant differences in performance requires a closer analysis of the meaning of "general" and "specific" categories.

On the whole, the subjects were quite successful in describing the indicated regions in the Venn diagrams. Some subjects avoided the confusion with the words "and" and "or" between boolean logic and English by separating their phrases with commas. It was surprising to see the use of negation in many of the descriptions. Subjects used negation in 80% of their descriptions when it was needed. This result is contrary to results mentioned previously by Vigil and by Wason. The results of this experiment indicate that it is the language used to express the concept which promotes errors, and not the misunderstanding of the concepts.

Despite this being a recognition experiment, there is evidence to suggest that users would be able to generate the required diagram from a natural language description of a query just as successfully. Therefore, Venn diagrams would be a more powerful and natural means of expressing boolean queries and should be used and tested in a keyword query interface in place of relying on post-coordinate boolean expressions. Possibly formal or natural language query systems could generate such diagrams in

response to a query for verification and subsequent modification by the user.

2.4. Keywords in Videotex

Keywords are used as a means of retrieval in many database and document retrieval systems. The following definitions were obtained from E.H. Brenner's "Indexing in Perspective" and are used by Ball and Gecsei in their discussion of keywords and keyword systems for videotex databases [Brenner 79] [Ball 81].

- *keyword* — grammatical element which conveys the significant meaning in a document. Word indicating a subject discussed in a document. In record oriented databases a keyword may be the value of an attribute, or an attribute-value pair.
- *index* — systematic guide to items contained in a collection. These items are represented by entries (e.g. keywords, descriptors) arranged in a known searchable order, such as alphabetical, chronological or numerical.
- *pre-coordinate indexing (processing)* — documents that enter the database are classified according to a fixed (authority) list of descriptors; this is usually done by qualified human indexers. On retrieval, the response (documents) is found in one particular place of the index. The term pre-coordinate indicates that keywords are combined (coordinated) into subject descriptors prior to the search process. Examples are back-of-the-book indexes or the fixed arrangement of menus in typical videotex databases.
- *post-coordinate indexing (processing)* — documents entering the system are described typically by a set of single words, determined by the author of the document (or by a computer algorithm). These words enter the index. Coordination of index words is performed at search time by forming logical products, sums and complements on the sets of documents found under individual index entries. The retrieval is controlled by user-supplied queries, often in interactive mode.

There are four major ways of accessing videotex databases: menus, keywords (pre-coordinate), alphabetic index, and printed directory. The menu approach is the most widely used access method, but studies suggest that even a crude pre-coordinate keyword system can reduce search time and the number of pages accessed as compared to the other three methods mentioned above [Stewart 80].

Many videotex systems permit direct access to known pages by letting the user specify a unique keyword for a given page as an alternative to traversing the menu structure. Variations of this scheme allow for different keyword dictionaries to be searched depending on the user's location in the database [Bochman 81]. The Danish videotex system also allows for alphabetic menu access by entering a keyword for a page as opposed to a numeric selection from a menu. This system permits the

combination of several keywords in one request, but they are interpreted as if they were individual requests. Despite having to use an awkward syntax, users in field trials preferred this type of access over menus [Orsnaes 82].

Pre-coordinate keyword systems are relatively easy to implement, but indexers must anticipate search patterns of users when constructing the keyword dictionaries. In a study done by Furnas et al. examining the potential performance of keyword information systems, it was found that random pairs of people used the same word for describing an object only 10-20% of the time [Furnas 82]. Furnas et al. state that the difficulties arise from a severe and fundamental lack of consensus in the language community on what to call things. They also found that experts do not describe things any better than others do. These results imply that pre-coordinate indexers do not have an easy task.

In post-coordinate systems indexers simply describe the content of the database rather than anticipate a user's search strategy. This allows the indexer or computer program to assign various descriptive keywords to a page independent of other pages, thus simplifying the indexing process at the expense of more complicated search methods. These query languages allow for dynamic generation of response information as opposed to following predetermined paths. User requests that return too many or too few documents can be modified until satisfactory results are produced.

Search strategies may be more complicated because the user may have to consider variants of potential keywords. In order to maximize the amount of information retrieved for a particular keyword, one may have to include its variants as in *computer, computing, micro-computer.* Online library catalog systems employing the post-coordinate keyword scheme use query languages with explicit and sometimes complicated syntax, usually based on boolean algebra and requiring the kind of logical thought that is known to be difficult [Wason 72].

Existing post-coordinate query languages have not taken advantage of graphics or pointing devices in order to simplify query specification. However, Venn diagrams seem to be a likely device to reduce the query specification problem in post-coordinate keyword systems. With Venn diagrams in mind, a post-coordinate keyword access system could be implemented as a more powerful interface to existing videotex systems. A prototype system has been developed that combines these features together with a graphics interface and a pointing device into an alternative access method for a videotex database.

2.5. Description of Keyword System for the UW 25th Anniversary Database

The main goal of this thesis was to develop an alternative keyword access method to a 3000 page videotex database located at the University of Waterloo. The database is stored on a DEC VAX 11/780 running Unix. Each page is stored as a file within a hierarchical file structure based on the existing menu access to the database.

Present post-coordinate keyword systems have command-based interfaces that involve query specification with boolean expressions. A more powerful interface that would combine modern methods of specifying keywords and handling results was desired. Features of Michard's GQL have been used to implement a prototype system, which involves selecting keywords from a fixed vocabulary and constructing post-coordinate queries with Venn diagrams.

The prototype interface was implemented with the emphasis on simple, useable features so that an initial working version could be produced. For a discussion of the interface design decisions made and possible alternatives, see Section 6.

The interface is composed of three positionally fixed or "tiled" windows as shown in Diagram 2.13, which remain displayed during an entire session. Each window is responsible for managing a certain logical aspect of the interface. One window is responsible for managing the list of keywords, another for presenting the Venn diagram, and the third for displaying the query result.

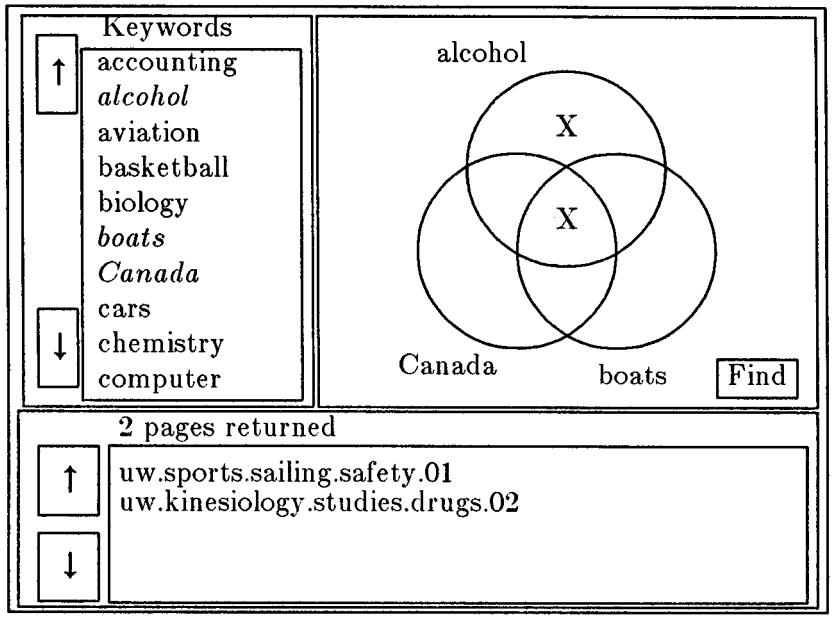


Diagram 2.13 Keyword system interface

The window responsible for the presentation of the keywords contains a list of alphabetically sorted keywords which can be scrolled by the use of soft buttons within the window. The Venn diagram is constructed dynamically by selecting words from the keyword list. By pointing to and selecting a keyword, the word becomes highlighted (by changing its colour) within the list, and a new circle is added to the Venn diagram with the word attached to it as a label. If three keywords have already been selected and the Venn diagram is full, then newly selected keywords are not added to the diagram. A specific keyword may be removed from the query by selecting it either from the list, or from the diagram. The corresponding word is removed from the Venn diagram and the word in the list is returned to its original colour.

In order to specify a query with a selected set of words, the user must point to and select one or more areas within the Venn diagram. Selected areas are indicated with a red "X" which may be toggled off by reselecting the corresponding area. Each time a word is added to or removed from the diagram, the Venn diagram is cleared of previously selected regions and redrawn.

When the user is satisfied with the composition of the query, the query processing is invoked by selecting the FIND soft button in the Venn window. This passes the query to the keyword database where the query is processed. The query result is then sent back to the keyword system's result window to be displayed.

The query result appears in the result window. At the top of the window there is an indication of the number of pages in the solution. The names of those pages matching the query are displayed in the result window. The result window can display up to five page names, and there are soft buttons, as in the keyword window, for scrolling through longer lists. By pointing to and selecting a page name in the result list, the user can display the particular page on an accompanying videotex terminal. When subsequent queries are invoked from the Venn diagram window, the result window is cleared to receive the next list of results.

Graphics and Workstation Considerations

3.1. Hardware Configuration

The keyword graphical query language was developed on an IBM PC/XT with 640K of RAM and a three-button Hawley mouse. The operating system used was Waterloo Port Version 2.3, a multi-process, message passing operating system [WMI 86b]. Sample programs are shown in the Port language which is similar to C; important points are explained as necessary.

The user interface is displayed on a Sony KX-1901A Trinitron component TV, and the user interacts with the system using a mouse and keyboard attached to the Port workstation. A NAPLPS (North American Presentation Level Protocol Syntax) intelligent decoder and 8088 co-processor board known as a *Quickpel* was mounted in the IBM XT and used to produce videotex graphics on the Sony monitor. The Quickpel has several features :

- 256 x 200 pixels
- ability to display 16 colours simultaneously from a possible 4096
- RGB or composite output
- 16K on-board RAM
- ability to run coprocessor programs independent of the host
- ability to manipulate the graphic display memory and colour map directly

The IBM PC/XT communicated with a DEC VAX 11/780 running Unix, which maintained the videotex pages and the keyword database. An Electrohome Integrated Videotex Terminal was connected to the VAX and was used as the output display device for the videotex pages.

Two monitors were needed for this application because the database contained pages that were encoded with the Telidon 699 standard, and these could not be displayed with the Quickpel board used for the user interface. This is because of the minor incompatibilities between the two standards used. Also, the interface would have been more involved and slower if both the interface and the pages were displayed on the same monitor because of the slow speed of NAPLPS drawing. See Diagram 3.1 for a layout of the configuration.

3.2. Software Configuration

3.2.1. Terminal I/O

During normal operation, a cursor is displayed on the monitor of the Port workstation under control of cursor keys and the mouse. However, the user interface for the keyword system was designed to be used with a mouse for pointing to and selecting items on the Sony display device. Unfortunately, Port only gives the cursor's position to an application window process when a key on the keyboard has been depressed and not

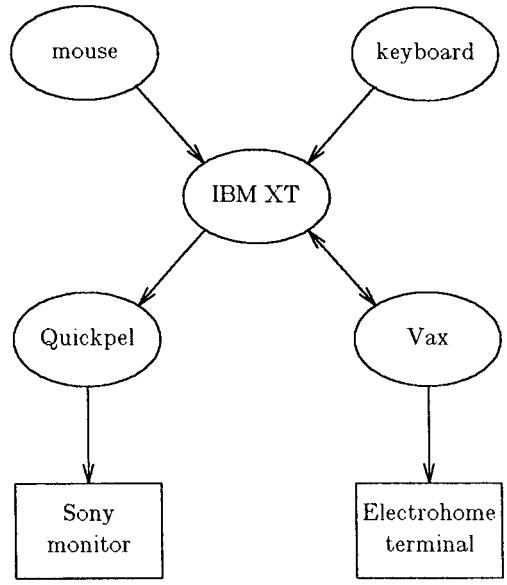


Diagram 3.1 Hardware Configuration

when the mouse is moving about. Therefore, it was necessary to take control of keyboard I/O from the Port operating system in order to control a cursor on the Sony monitor. This involved taking over monitoring the serial line connected to the keyboard and mouse from the Port process normally responsible. The lines of code in Diagram 3.2 are necessary to take over keyboard I/O and later give it back for normal operation after the application has terminated.

Note that when taking over the terminal, the field *WHICH* in the IO_request template must be set to 1 so that the system knows which window number is being taken over. Care must also be taken to have a reply message variable of type *unsigned[1]* when giving control back to the *terminal master* process. The action of taking over terminal I/O is undocumented and not considered normal procedure in Port application development.

3.2.2. Quickpel board

As mentioned previously, a separate monitor was used for the user interface because of the incompatibility between NAPLPS and Telidon 699. The Quickpel board mentioned in Section 2.1 is a NAPLPS decoder; therefore, its graphics capabilities were used rather than writing graphics routines for the video memory on the board. The Quickpel is a peripheral device normally used with an IBM PC running DOS. However, the IBM PC used for the keyword system was running the Port operating system and as a result, there was a problem with the drivers for the Quickpel. Port uses interrupt line 2 for a network expansion card. Since our Port workstation was configured as a stand-alone system, interrupts on line 2 would be ignored. The Quickpel drivers also use interrupt line 2 for an

```
import(IO_request, IO_requests, Server_names,...)
()
terminal_request : IO_request
junk             : unsigned[1]
{
      .
      .
      .
REQUEST[terminal_request] = TAKE_OVER_TERMINAL;
WHICH[terminal_request]   = 1;
send( terminal_request, terminal_request,
            Get_registered_id( TERMINAL_MASTER ));
      .
      .
      .
REQUEST[terminal_request] = TAKE_BACK_TERMINAL;
send(terminal_request,junk,Get_registered_id(TERMINAL_MASTER));
Destroy(My_id);
}
```

Diagram 3.2 Keyboard I/O

interrupt driven keyboard interface. Unfortunately, the drivers for the board were only available as object code and could not be altered even slightly. Therefore, a driver written in Port was required to use the board. The Port driver operates in a polling manner as opposed to being interrupt driven. However, this was not a problem because of the multi-process structure used. It was unnecessary to have a complete driver for the board (coprocessor software loading, remote database access, receiving data from the board) because only NAPLPS commands were sent to it.

The Quickpel has its own multi-tasking executive running in the board and has a *PLPS* process that waits for input on the serial line. (Refer to Electrohome's user manual on the Quickpel for further details [Electrohome 84].) Input received by the *PLPS* process is decoded into NAPLPS commands, and output is written to the display memory of the visible screen. The Port work station transmits and receives data from the Quickpel through a bidirectional, eight bit parallel port. The port interface consists of four registers which are located in Port's address space relative to a base address set on the Quickpel board. The base address is set to hexadecimal '8380', henceforth written 8380H to follow Electrohome's notation and coded as $8380 in the Port language. The four registers are numbered 0 through 3 (e.g. 8380H, 8381H, 8382H and 8383H). Register 0 is a bidirectional, eight bit data port. To this register one can pass NAPLPS codes, and special commands to be interpreted by the board (e.g. cold start, display graphics cursor, etc.). Received from this register are locally echoed data, transmitted unprotected field contents, or the xon/xoff protocol. Register 2 is a uni-directional, eight bit status port that is read only and returns the present status of the Intel 8255A programmable peripheral interface microprocessor on the Quickpel

board. When bit 7 of register 2 is set to 1, this indicates that the Quickpel board has accepted the previous byte and is ready to receive another character. The code in Diagram 3.3 is the *Send* function used in the application to send data to the Quickpel board.

```
\ DESC: This function waits until the ready to
\       receive bit is set and that an Xoff signal is not present
\       before transmitting the byte to the quickpel board.
( ch : unsigned[8] )
{
repeat
 if (IO_in($8382) > 127)
   break;
repeat
 if (IO_in($8380) != $13)
  break;
IO_out($8380, ch);
}
```

Diagram 3.3 Send *function*

The Quickpel board has a 2K buffer for input. An xon/xoff protocol is observed in the *Send* function because the 2K input buffer may become full, although this is unlikely since only individual NAPLPS commands are sent to the board. However, for robustness, register 0 is checked to make sure that an xoff byte has not been sent from the board before transmitting a character to it.

Register 3 is an eight bit control port that is used for setting and clearing interrupt enable bits associated with the 8255A. In order to initialize the Quickpel board and put it into a known stable state, one must send the magic byte C4H† to the 8255A in register 3. A non-selective reset is also sent (1FH) to reset NAPLPS to its default values. Diagram 3.4 shows the *Qpel_init* function used to initialize the Quickpel board.

```
\ DESC: This function initializes the quickpel board.
()
{
  IO_out($8383, $C4);    magic byte
  Send($1F);             non-selective reset
}
```

Diagram 3.4 Qpel_init *function*

---

† This magic byte was undocumented and known only to gurus.

3.2.3. The Cursor

NAPLPS systems are not well-suited for mouse controlled cursors that move about the entire screen. As a result, it was difficult to get a cursor working for the keyword system as well as extract the necessary information on how to attempt it from the CSA standard for NAPLPS [CSA 83]. The following definitions obtained from the CSA standard will be useful in discussing features of cursors in the videotex environment.

Cursor — a logical indicator (having character field dimensions) of the screen position at which the next character is to be deposited. This position may or may not be marked by a cursor symbol.

Drawing point — a logical indicator of the position at which the next geometric graphic primitive will commence execution. This is normally not marked by a drawing point symbol.

Logical picture element (logical pel) — a geometric construct associated with the drawing point whose size determines the stroke width of the graphics primitives.

NAPLPS systems contain a text cursor and a graphics point, each of which could be used as a general cursor for an application such as the keyword system. With the Quickpel, it is also possible to display a graphics cursor. The graphics cursor consists of an exclusive-or cross-hair centered on the current graphics drawing point. The cursor's height is 20 pixels plus the height of the logical pel and the width is 20 pixels plus the width of the logical pel. At the intersection of the cross-hair is a hole which is the size of the logical pel. Initially the graphics cursor is off. The command (1BH,3EH) will turn it on and the command (1BH,3FH) will turn it off again. There are problems associated with using each type of cursor as the general cursor for the keyword system.

In order to have the mouse drive the text cursor, the following steps must be taken. By default the entire screen is considered a protected field. One must use the unprotect command from the C1 code set (1BH,5FH) to unprotect the entire screen so the text cursor will move about on the screen when given cursor movement commands. By default the text cursor is invisible, and one can set it to visible and solid (1BH,5CH) or flashing (1BH,5BH). To control the movement of the text cursor, there are commands that can be used from the C0 control set that move the text cursor left, right, up, down, and set its position on the display device [CSA 83 (p.97)]. Alternatively, the following bytes could be sent to the keyboard port (register 1) of the Quickpel as opposed to the NAPLPS command port (register 0):

up : C1H *down* : C2H *left* : C3H *right* : C4H

However, there are no commands that can be sent to the keyboard port to position the text cursor at a particular location. The text cursor is moved the dimension of a character upon receipt of either type of command and comes in four different shapes, each of which can be either solid or flashing. The four shapes are : underscore, block, cross-hair, and

custom. The definition of the shape of the custom cursor symbol is implementation-dependent and Electrohome uses an outlined box for the Quickpel.

There are two major problems with using the text cursor. The most annoying is that it is not possible to display a string of text on the screen using proportional spacing for the characters when the entire screen is unprotected, as it must be to give the cursor the freedom to move over the entire area. Also, it is not possible to vary the precision or number of pixels the cursor moves.

It would be possible to use the drawing point as a general cursor but it would require modifying the logical pel size from a single pixel to a size more readily visible, such as 10 x 10 pixels. As a consequence of this, one would have to toggle the size of the logical pel back and forth when a geometric figure had to be drawn, otherwise the geometric figures would have excessively large boundaries. Also, whenever a geometric figure was to be drawn, the current location of the drawing point would have to be saved and later restored because picture drawing instructions alter the location of the drawing point. Modification of the logical pel does not alter the style in which text is displayed.

Another alternative is to use the graphics cursor supplied by the Quickpel as the general cursor for the keyword system. There are no explicit commands for moving the graphics cursor up, down, left and right as with the text cursor. However, one can move this cursor to any pixel location on the screen by using the *Point* geometric drawing primitive as would be needed for the drawing point style of cursor described above. The graphics cross-hair cursor does not require modification of the logical pel size, but its location is altered by picture drawing instructions. However, it is always shown at the second most recent position of the drawing point because of the character look ahead options available with NAPLPS.

The graphics cursor has been used in the keyword system because it was desirable to have proportional text displayed within the windows. However, the cursor does not behave as well as would be desired. It lags behind the movement of the mouse for gross mouse movements and flickers considerably. This is because of the continual need to set the drawing point for each movement. To compensate for this, a variable cursor stepsize has been implemented so that gross mouse movements will cause the cursor to cover a larger distance. Slower mouse movement keeps the fine stepsize for precise positioning. This technique has helped somewhat, but further improvements would be desirable.

3.2.4. Implementation of graphics primitives

Only three of the basic geometric drawing primitives and some attribute control functions available in NAPLPS were needed for constructing the keyword interface. The picture drawing instructions used were: *Point*, *Arc*, and *Rectangle*. The *Point* primitive is needed to set the location of the drawing point for the cursor and before drawing geometric figures if the relative displacement option is used. Diagram 3.5 shows the

NAPLPS format for the four byte *Set Point (Absolute, Invisible)* command.

	b8	b7	b6	b5	b4	b3	b2	b1
byte 1	X	0	1	0	0	1	0	0
byte 2	X	1	±	x8	x7	±	y8	y7
byte 3	X	1	x6	x5	x4	y6	y5	y4
byte 4	X	1	x3	x2	x1	y3	y2	y1

Diagram 3.5 Command format

The eighth bit in all four bytes, marked with an *X* in the diagram, is used as a parity bit or an extension to another code table. For this application the seven bit coded character set was used so the eighth bit would be set to one for all bytes. The first byte is the command byte for the instruction. The next three bytes contain the x,y coordinates at which to set the drawing point. If the seventh bit in a byte is zero, this indicates that the byte is a command, otherwise the byte is data. The x,y coordinates must be encoded into the data bytes in the order indicated in the diagram. This byte sequence is typical of the format used for commands in NAPLPS.

The code from the *Set_Point* function in Diagram 3.6 shows how the *Send* function is used to implement the *Set_Point (Absolute, Invisible)* command in the keyword system.

```
( x : unsigned[8]
  y : unsigned[8] )

set_point : unsigned[8]
data1     : unsigned[8]
data2     : unsigned[8]
data3     : unsigned[8]

{
  data1 = 192 + (x & 192)>>3 + (y & 192)>>6;
  data2 = 192 + (x & 56) + (y & 56)>>3 ;
  data3 = 192 + (x & 7)<<3 + (y & 7);
  set_point = $A4;

  Send(set_point );
  Send(data1     );
  Send(data2     );
  Send(data3     );
}
```

Diagram 3.6 Set_Point *function*

Other functions were implemented to draw circles, rectangles, text, and set the current drawing colour. Rectangles can be drawn either outlined or filled with a colour. To draw text is simply a matter of setting the drawing point and sending the characters in the string.

Software Development Environment

4.1. Waterloo Port

Waterloo Port, a network operating system for personal computers marketed by Waterloo Microsystems Inc., encourages the use of multiple inter-communicating processes for applications programming. Port runs on IBM-PC and compatible microcomputers and has a powerful networking capability. Port also has an interesting iconic user interface employing a *room* metaphor as opposed to the *desktop* metaphor common to systems such as the Apple Macintosh and Xerox Star. Interaction with Port is by means of a mouse for pointing and selecting items to execute.

Port is not only an operating system, but also a programming language. The Port operating system and programming language were developed in 1980 by the Software Portability Group at the University of Waterloo. The Port programming language is closely related to the C programming language: it has an expression syntax similar to that of C but is quite different from C in many respects. Port makes use of a tree-structured file system and source programs are also tree-structured [Cargill 79]. For complete details of the Port programming language refer to *Waterloo Port's Port Development System Manual* [WMI 86a].

A *program* is a sequence of instructions that can exist as source code, load module and executable module. A program does not *run* or *execute*, but is simply a sequence of instructions to be followed. A *process* is an active invocation of a program. At any point in time, a process has a state (blocked, ready, active) and is said to *execute* a program. It is possible, and often desirable, to have more than one process executing the same program [Malcolm 85].

Multiple process systems are useful in environments where multiple tasks need to be accomplished concurrently. Such a situation will become increasingly common in public databases such as those provided by videotex. Multi-process systems can also yield simple solutions to non-concurrent problems as well.

In a Port application program there is usually one process which is responsible for creating all of the other processes needed for the application. Any process can create another process using the system function

new_process_id = Create(pathname, priority);

where *pathname* is a string that specifies the file in which the new process' executable code image is stored, and *priority* specifies the priority of the new process on a scale of 0 (low) to 255 (high). A process can be referenced by other processes using its assigned unique process identifier. A process may similarly be destroyed by another process using the system function

Destroy(process_id);

The system library external variables *Creator's_id, My_id,* and *My_priority* exist so that a process may know who its creator was, what its process identifier is, and what its priority is. These values may be passed on to other processes or functions.

Processes are always in one of three states. A process is considered *ready* when it is able to perform the next action of its program. The process currently executing is considered *active.* If a process is not ready, then it is *blocked.* A process becomes blocked when it does any one of the following :

- □ sends a message to another process
- □ waits to receive a message from another process
- □ delays for a specified period of time
- □ requests a system service such as creation

When a process becomes ready, it is placed in the *ready queue.* Processes in the ready queue are ordered from highest to lowest priority, and within each priority level, by their times of arrival. When the processor becomes available, the process at the front of the ready queue becomes *active.* Active processes relinquish control of the processor when they block or when a higher priority process becomes ready (for example, as a result of an external device interrupt). When active processes are preempted they are returned to the ready queue and will be resumed when they reach the front of the queue. A process that is blocked on a message-passing operation may become ready when the message-passing transaction is complete. Processes communicate with each other using the system primitives:

send(message, reply_message, receiver_id);
receive(message, sender_id);
reply(reply_message, sender_id);

There are four different modes of blocking that result from message passing. When a process executes the *send* operation, it becomes *send_blocked* until the receiving process receives the message, at which point it becomes *reply_blocked.* When a process executes the *receive* operation it becomes *receive_blocked* if the sending process has not already sent the message. If a process wants to receive messages from an unspecified process it can execute the system primitive :

sender_id = receive_any(message);

If there are no messages waiting for a process which executes this operation, then the process becomes *receive_any_blocked* until a process sends a message to it. A process does not block when executing the *reply* operation.

So as not to be overwhelmed by the complexity of programming large multiprocess systems, it is convenient for the programmer to use metaphors to aid in the design and understanding of the program structure.

The attribution of human characteristics to processes is known as anthropomorphic programming, and it is a proven technique for building systems that work using multitask structuring and message passing schemes to simplify the analysis of complex systems [Booth 84].

Some examples of anthropomorphised process structures commonly used in Port applications are :

- *proprietor*
- *administrator*
- *worker*
- *client*
- *courier*

These models were developed based on behavior patterns of individuals within organizations and how problems are solved by organizations in society.

A *proprietor* is a process that is considered to *own* a resource and has exclusive access to it. This type of process implements operations on the resource, in a first-come-first-served manner. At most one client is reply_blocked at any one time, but many clients may be send_blocked. A typical example would be a process that was responsible for access to a piece of hardware such as the display device. Diagram 4.1 shows how processes interact with a proprietor. In each of the process diagrams or blocking graphs, the arrow indicates the direction of the *send* primitive.

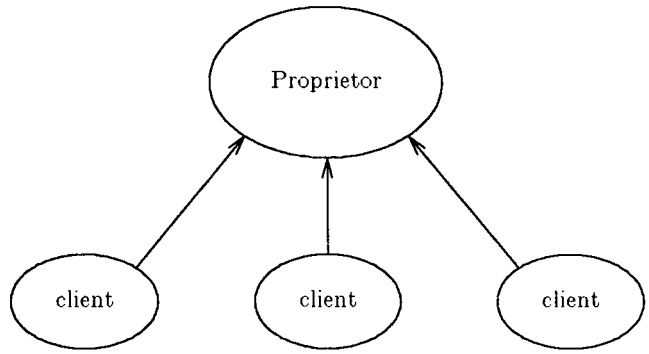


Diagram 4.1 Proprietor

An *administrator* process does not become tied up processing a single request as a *proprietor* process does. It has *worker* processes which perform the required service on its behalf so that it may serve many *clients* at once. Requests are not necessarily handled first-come-first-served, and more than one client or worker can be reply_blocked at a time. Implementation of an administrator is more complex than that of a proprietor because the administrator has to maintain queues of workers and client processes and manage them effectively. However, the use of the administrator increases the amount of parallelism in the design because it is only busy when there is work to delegate, otherwise it is available to process additional client requests.

A *worker* is usually a special purpose process that performs a certain action for an administrator. A worker sends a message to its administrator when it is ready to do some work. The worker will remain reply_blocked until the administrator later replies with a message indicating what work is to be done.

A *client* is usually a process in the application that requires certain tasks to be carried out on its behalf. It usually sends requests to an administrator or a proprietor, and may itself be a proprietor or administrator for some other resources. Diagram 4.2 shows an administrator with three worker processes and three clients.

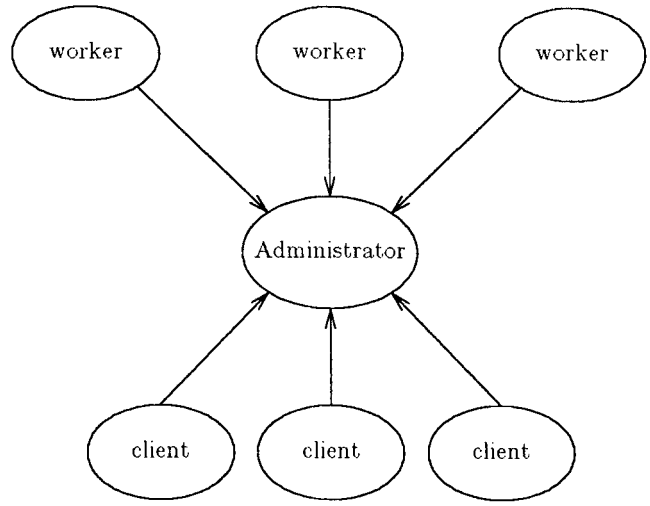


Diagram 4.2 Administrator

A *courier* is a special process that is used to pass messages on behalf of a process that does not want to block waiting for a reply. This type of process can be used by an administrator when it needs to communicate with another process outside of those which it is supervising. Diagram 4.3 shows a scenario where a courier is used to pass messages from administrator A to administrator B. Diagram 4.4 shows some sample code that could be used to implement the courier in this example. It is assumed that the courier is created by administrator A and that administrator A replies to the courier the process identifier of administrator B so that the courier knows the destination of the message.

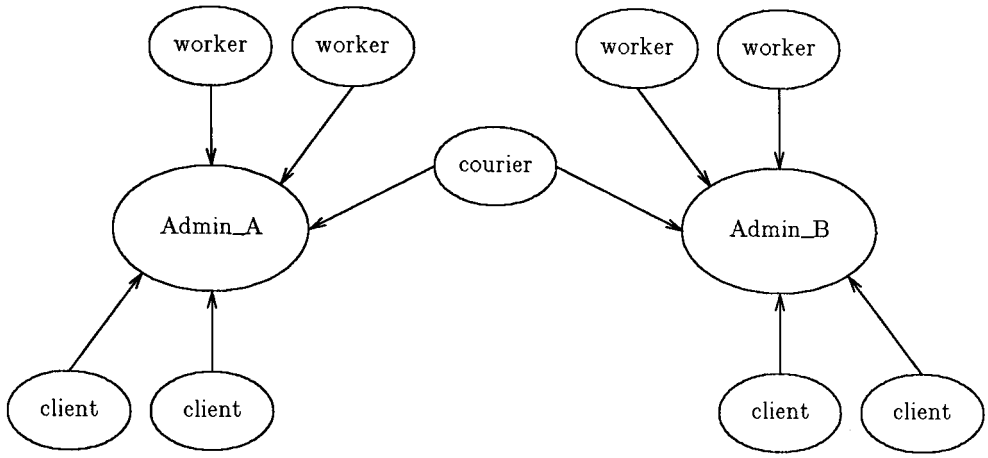


Diagram 4.3 Administrators with Courier

```
\
\ sample code for a courier
\
import( Message_type, Courier_Requests )
()
  admin_B : pid
  msg     : Message_type
{
  send( WHOIS_TARGET, admin_B, Creator );

  repeat {
     send( READY, msg, Creator );
     send( msg, msg, admin_B );
  }
}
```

Diagram 4.4 Code for a Courier

4.2. Process structuring

Developing an application using multiple processes requires attention to problems in combining the various processes that do not exist in a single process application. These problems include process structuring, deadlock, prioritization, inter-process communication and synchronization.

Deciding how to structure a problem using several processes is not trivial, even in an anthropomorphic model. Usually proprietors are assigned to handle I/O for serial lines, keyboard input and display devices. Proprietors are generally fairly easy to code, but deciding how to structure the rest of the application program can involve significant effort. Lack of experience in developing applications in this type of environment may result in process structures that are hard to debug. Unfortunately,

there is limited literature that discusses anthropomorphic programming or process structuring *at all*. Therefore, there are few guidelines or general rules to follow.

A major concern when developing the keyword system was to avoid deadlock among the various processes involved. Deadlock can generally be avoided by examining a blocking graph of the process structure. In a blocking graph, all processes are drawn with an arrow from the sending process to the receiving process. If a cycle exists in the graph, then there is potential for deadlock. Couriers can be used as previously mentioned to break such cycles.

Although a process in Port may be assigned a priority from 0 to 255, the development system manual recommends that application programs should only have priorities between 0 and 127 so as to not interfere with the operating system. There are no additional guidelines for selecting priorities for processes. The performance of an application can be severely affected by changing the priorities of the processes involved. An application programmer attempting to employ the abstract process models previously discussed within a design will undoubtedly have to decide on their priorities. How should the priorities of administrators, workers, couriers, proprietors, etc. be related? In real time systems (e.g. controlling a nuclear power plant), certain events require "urgent" attention, but user application programs (e.g. a spread sheet) seldom have these constraints. In general, "urgent" processes should be given higher priority over "less urgent" ones, but often it is not clear in user application programs whether urgent processes are those with more work to do (e.g. workers) or those that manipulate user-perceptible system parameters (e.g. proprietors).

Careless use of communication primitives can easily lead to trouble. A common error during initial system development is forgetting to reply to a process after having received a message from it or after having serviced a request for it. This type of error causes the sending process to block indefinitely. Another common error occurs when a process sends a message of a different type than the receiving process is expecting. This causes the sending process to be destroyed and a system message is presented to the programmer. Debugging applications is quite difficult because of the non-sequential nature of the execution of the processes. However, this may be true in general of systems that must interact with many external devices and service them appropriately.

Application development with multiple processes and process structuring is a non-trivial task requiring a thorough understanding of the abstract models and some experience. Software support should be provided for those programmers unfamiliar with this area, but wish to exploit the design possibilities available for user application development in a multiprocess environment. Automatic restrictions on structuring of control flow is common in high level programming languages because it enforces what is considered to be good programming methodology. Enforcement usually consists of providing a compiler that rejects

unacceptable control structures and checks for parameter passing errors. Although one could write a well structured assembler program, one may prefer to use a higher level language such as PASCAL, since the compiler will automatically reduce the chance of simple errors in control flow. A similar need exists for restricting high level process structures and control of message-passing to allow for robust structuring of processes.

4.3. Process manager

A structure known as the *process manager* can be used to monitor and control processes [Raymond 86]. By restricting the process environment, it simplifies the programming involved in developing an application. This was a reasonable approach to follow for this project, since there were no excessive performance requirements and a flexible development would be beneficial. The process manager was first successfully used in developing a structure editor, which is a graphical tool for manipulating files in an information system. The process manager was used in the development of the keyword system to test the structuring model further, and to allow for greater ease in combining the structure editor and the keyword system in the future.

The main characteristics of the process manager are as follows :

- a restricted set of subordinates known as *managers*
- a non-blocking communication scheme
- monitoring of communication between managers by using couriers
- detection of "runaway" managers and restriction of their communication
- automatic creation and destruction of manager processes involved in the application

Managers resemble proprietors in that they are responsible for some resource or set of activities, but managers differ from proprietors because they are able to communicate with other managers without blocking while servicing a request. Most managers consist of two parts: a main routine that obtains requests from other managers, and a set of functions that perform the requested actions. The function chosen to service the request may generate a request to another manager as well. Diagram 4.5 shows a typical main routine for a manager that obtains requests and services them with the appropriate functions.

The major difference with respect to process structuring with the process manager is the use of a non-blocking "send" function: *Request*. This deviation from the synchronous communication scheme employed in Port inherits the problems associated with asynchronous communication in real time systems such as queuing of messages and management of finite resources. However, systems that do not have real time constraints or need processes to synchronize temporally could use this communication scheme. The process manager only guarantees that requests are eventually delivered and will be serviced in the order in which they were made. For a more indepth discussion of the process manager and communication considerations see pages 5-9 of Raymond [Raymond 86].

```
import( Request_Types )
()
 request : Request_Type
{
   Initialize();

   repeat {
     request = Get_Request();
     select(request) {
        case REQUEST_A : Service_A();
        case REQUEST_B : Service_B();
        case REQUEST_C : Service_C();
        case REQUEST_D : Service_D();
     }
   }
}
```

Diagram 4.5 Manager's main routine

Managers can communicate with other managers by using the request:

Request(ACTIVITY, MANAGER_NAME);

The parameters passed in the primitive are constants or manifest values as opposed to variables. (Within the Port language, manifest values are in upper case.) ACTIVITY is a manifest constant indicating a service desired from the list that the manager MANAGER_NAME offers. The execution of the function actually sends a message to the process manager, who then allocates a courier for the message. Thus the process manager monitors communication between managers. *Get_Request* is the only other communication function: this blocking function behaves similarly to the *receive_any* primitive in Port.

Functions servicing a request do not use Get_Request. This rule is followed so that managers will only block in their main routines; this makes the process communication more easily understood. Activity within a manager happens sequentially whereas activity between managers happens concurrently. Hence the question "should I use a manager or a function here?" can be answered by answering the simpler question "should this activity happen in parallel with other activities?"

Since managers do not use the Port message passing primitives, the process manager can monitor the communication between the various managers within the application. For each request, the process manager allocates a courier from a list of maintained couriers. Therefore managers can communicate with each other without having to worry about simple deadlock. Because the process manager is allocating the couriers to facilitate the message passing, a manager that generates requests faster than they can be serviced can be detected and restrained. To further aid development, the requests that each manager services are manifest constants from a central file that can be checked at compile time when

developing a new manager process. An unknown request received by a manager at run-time will be ignored by simply falling through the select statement and will therefore not cause any major problems. This feature also facilitates incremental development of the process structure by the use of program stubs.

The first activity of the process manager is to create its couriers and then the other managers in the application. The process manager also automatically assigns highest priority to itself, the couriers next, and the other managers have equal and lowest priority. When the application is finished, one of the managers will make a request to the process manager to destroy the application. The process manager will then invoke an application specific destruction module to destroy the managers in an orderly fashion.

Design of the Keyword System

5.1. Overview

The emphasis of this project was to develop a powerful and simple keyword interface for a videotex database; therefore interaction with a menu system or other browsing methods were not considered. The size and implementation of the keyword database was not a major concern. The prototype system was tested on the the 25th Anniversary Videotex Database stored on a DEC VAX 11/780 running Unix at the University of Waterloo. The database is stable and unchanging, so the keyword dictionary could be simply generated once. This eliminated many problems associated with maintaining the keyword dictionary in a volatile environment. Given that this is only a prototype system, many features were implemented in a simple fashion.

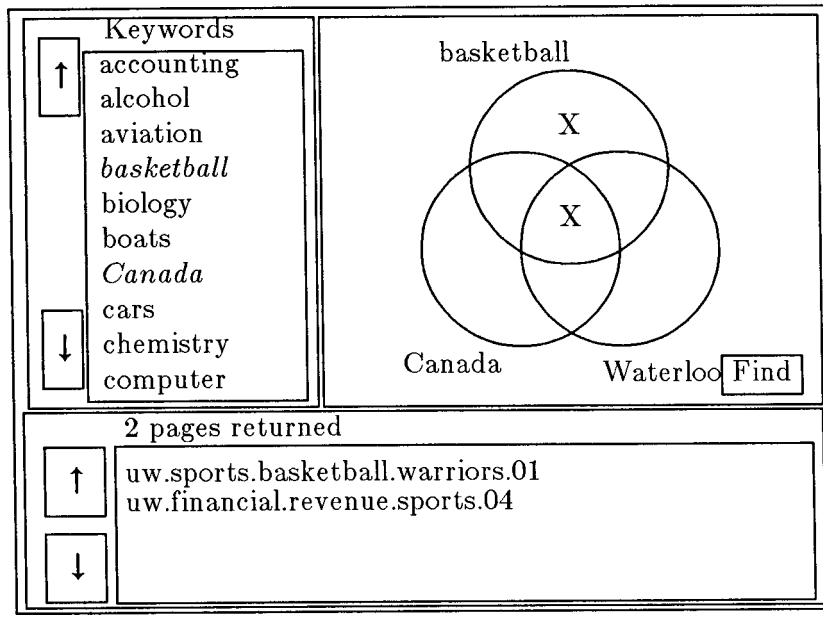


Diagram 5.1 User Interface

Users do not enter keywords; instead they are selected from a list of valid words. This strategy reduces input time and error.

In order to handle multiple page matches for a query, the system required a result window for displaying the path names of those pages found. The result window initially returns a number indicating to the user how many pages were found and then the page names. For queries that return too many pages the user may continue, without examining any of the pages, by refining the query to be more restrictive. This standard method was considered superior to one in which all matching pages

are displayed regardless of the number returned.

Since the keyword list comprises valid keywords from the database, there will always be a match in the case of a single keyword selection. This is not true for systems where keywords may be input by the user. When a multi-keyword query fails, a zero is returned in the result window indicating that no pages were found. The only option for the user is to modify the query to make it less restrictive.

When the Venn diagram changes due to an addition or deletion of a keyword (see Section 2.5), it is not clear what to do with previously selected regions because the regions of intersection change when adding or deleting a circle. The user's previous work in selecting regions should probably not be thrown away and a reasonable approach would keep all selected regions allowing the user to remove undesirable ones. However, because it was the easiest to implement for the prototype system, the diagram is cleared of selected regions and completely redrawn in the case of a deletion. Also for ease of implementation, the circles are drawn in the order of left, right and above, and are always drawn in that order. If the left circle of a two circle diagram is removed, the right circle moves to, and replaces the left circle. A more intelligent scheme would leave the circles in their respective positions so as not to reconfigure the diagram each time and disorient the user.

The prototype system does not include a feature like Michard's MEMOR to encapsulate queries into a single circle to accommodate complex queries with more than three keywords. This feature would be advantageous, but was not included because of the added complexity and limited time for development. It was also speculated that a manageable number of pages could be found by specifying a maximum of three keywords and the user could scan through the list of pages returned in order to find the desired page(s).

The page names that are returned in the result window correspond to the Unix filenames for the pages. These names were adequate for development work. For a further discussion of this and other points, see Section 6.2.2 which describes alternatives for an improved system.

5.2. Process structure implemented

The process manager model, as discussed in Section 4.3, was used to control the process structuring and to simplify the development of the application. The user interface as described in Section 2.5 and shown in Diagram 5.1 was modularized into the managers shown in Diagram 5.2. The arrows in Diagram 5.2 show the paths of potential requests in the keyword system.

The activity of the keyword system is governed by the user's input from the mouse and the keyboard. Serial input to the system from the mouse and the keyboard are dealt with by the *Kybd_in* manager. The *Kybd_in* manager is similar to a proprietor except that it only generates requests and services none.

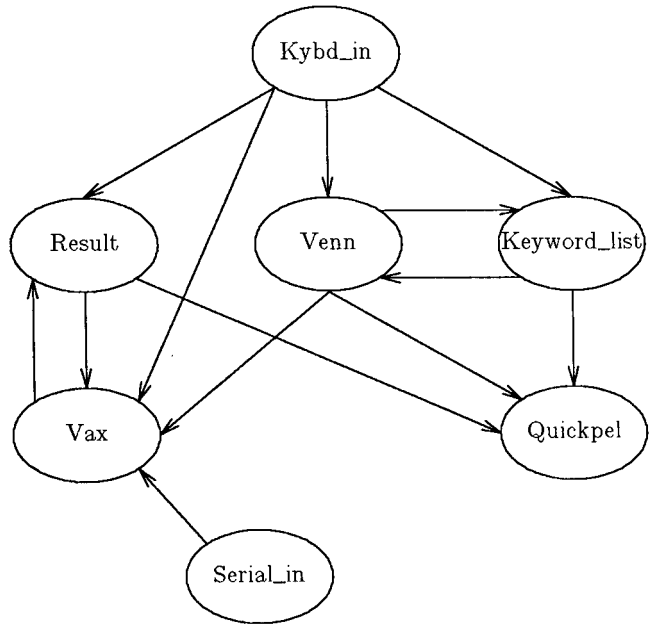


Diagram 5.2 Manager communication

There are three window managers (*Keyword_list*, *Venn*, *Result*) to manage the three windows in the interface and a display manager (*Quickpel*) which controls all output to the display device. Each window manager initializes by obtaining its window coordinates from a file and making a request to *Quickpel* to draw the initial window. Once the system has been initialized, each window manager must be able to handle a request from *Kybd_in*, determine if and where in the window a selection was made, and handle it appropriately. Each of the window managers evaluates requests to determine if they are applicable to the manager's associated window. If the request is applicable, then it is serviced, usually involving an update to the window. All modifications to windows are made by making a request to *Quickpel*, which is a proprietor controlling the display device.

Communication with the remote database is handled by two managers. The *Vax* manager is the main process responsible for the communication protocol between the keyword system and the remote database system. The *Serial_in* manager is really a worker process for *Vax* and only gathers input from the communications line and passes it on to *Vax* for processing. We now turn to consider each manager in detail.

5.2.1. Kybd_in

Kybd_in handles all serial input from the mouse and the keyboard attached to the IBM PC/XT Port workstation. In the current implementation, the only keyboard input recognized is the ESC key which is used to terminate the application. When this key is detected, a request is sent to *Vax* to terminate the session with the remote database system and to notify the process manager that the application should be destroyed.

Due to hardware restrictions discussed in Section 3.2.3, *Kybd_in* maintains the current cursor position on the display device as determined by the mouse. When the select button is depressed on the mouse, a request is made to the three window managers specifying the current cursor position. It is the responsibility of each window manager to determine if the selection was made in its window, and if so, how it should be serviced. Diagram 5.3 shows the main module for *Kybd_in*.

```
import( Characters, Process_List, Process_Requests )
()
  ch   : char
{
    repeat {
      ch = Get_A_Key();
      select(ch)
        {
         case  EXIT          : Request( TERMINATE, VAX );
         case  SELECT_M : Select();
        }
    }
}
```

Diagram 5.3 Kybd_in main routine

5.2.2. Quickpel

Quickpel handles all requests to produce visible effects on the display device. Each such request specifies the type of graphical object to be constructed and its (x,y) position in the display area. Hence, *Quickpel* essentially defines a small, high-level graphics package that interfaces to the board's NAPLPS capability. *Quickpel* services 18 different requests varying from drawing a single character to drawing the initial layout for a window. Diagram 5.4 shows the main module for *Quickpel*.

```
import( Dimensions, Process_Requests, Qpel_Manifests )
()
  request : unsigned
{

Init();
Get_initial_coords();
Clear_Screen();
Draw_Screen();

repeat
  {
  request = Get_Request();
  select(request)
    {
      case ADD_CIRCLE            : Circle( HIGHLITE );
      case DELETE_CIRCLE         : Circle( ERASE );
      case DRAW_WINDOW_V         : Draw_Window_V();
      case DRAW_WINDOW_KW        : Draw_Window_KW();
      case DRAW_INIT_RES         : Draw_Window_R();
      case DRAW_INTER_RES        : Draw_Inter_Res();
      case FLASH_COMMIT          : Flash_commit();
      case ADD_MARK              : Draw_char( HIGHLITE );
      case DELETE_MARK           : Draw_char( ERASE );
      case DRAW_STRING           : Draw_string( NORMAL );
      case ERASE_STRING          : Draw_string( ERASE );
      case HIGHLITE_STRING       : Draw_string( HIGHLITE );
      case WORKING_MESSAGE       : Display_Working();
      case CLEAR_KW_LIST         : Clear_kw_list();
      case CLEAR_NAME_LIST       : Clear_name_list();
      case FIX_CURSOR            : Fix_cursor();
      case CLEAR_LINE            : Clear_line();
      case CLEAR_SCREEN          : Clear_Screen();
    }
  }
}
```

Diagram 5.4 Quickel main routine

5.2.3. Keyword_list

Keyword_list manages the window that contains the keywords. When a keyword is selected from the keyword list, a request is made of *Venn* to display the word and a circle in the Venn diagram. *Keyword_list* initially reads the file containing its window coordinates and then reads a file of alphabetically ordered keywords to be stored in an array. The name of the file containing the keywords is specified when the keyword system is invoked from Port, and is passed to *Keyword_list* by the process manager. For the prototype system, a maximum of thirty keywords was permitted. After the keywords have been retrieved, a request is

made of *Quickpel* to draw its window. The keywords are then displayed in the window by successive requests of *Quickpel*.

Keyword_list scrolls the list of keywords when either the up or down scroll button is selected in the window. The current implementation uses a scroll height equal to the window height (ten words). When a word is selected, a request is made of *Venn* to display the word with an accompanying circle. Selected words are also highlighted in the keyword list. If a highlighted keyword is selected, then a request is made of *Venn* to delete that word from the diagram, and the word is restored to its original colour. For a further discussion of the implementation and improvements for this window, see Section 6.2.1. Diagram 5.5 shows the main module for *Keyword_list*.

```
import( Process_Requests, Process_List )
()
  request : unsigned
{

Get_initial_coords();
Get_Keywords();
Request( DRAW_WINDOW_KW, QPEL );
Print_list();

   repeat {
        request = Get_Request();
        select( request ) {
           case SELECT         : Check_cursor_position();
           case REMOVE_WORD : Change_word_to_normal();
        }
   }
}
```

Diagram 5.5 Keyword_list main routine

5.2.4. Venn

Venn is responsible for managing the construction of the Venn diagram, and the generation of the query when the diagram is complete. Queries are constructed by selecting keywords from the keyword list, resulting in a request to *Venn*. *Venn* stores the keywords in an array for easy manipulation and modification to the diagram when words are selected for deletion.

The selected regions of the Venn diagram are maintained with a status byte. Each bit in the status byte corresponds to an area in the Venn diagram. When a region of the diagram is selected, the appropriate status bit is set accordingly.

When the FIND button is selected, a query is generated based on the status byte and the number of keywords in the diagram. A request is then made of *Vax* to transmit the query to the remote database. A query has the format :

$$< \# \text{ keyword } \{ \text{keywords} \} \text{ query } >$$

where # is the number of keywords in the query, the keywords are given separated by spaces, and the query is a list of disjuncts representing boolean expressions for the areas selected in the diagram. The boolean expression consists of boolean connectives and symbolic names (numbers) representing the keywords for compactness. For example, a query with three keywords would look like :

3 Canada drugs computer (1&~2&3) (1&2&3)

When a request is made of *Vax* to transmit the query, another request is made of *Result* to indicate that a query is in progress, as shown in Diagram 5.6.

```
import( Process_List, Process_Requests )
()
  request   : unsigned
{
Get_initial_coords();
Set_expression_list();
Request( DRAW_WINDOW_V, QPEL );

repeat {
    request = Get_Request();
    if( request == SELECT )
      Check_cursor_position();
    else  Service_kw_mgr( request );
  }
}
```

Diagram 5.6 Venn main routine

5.2.5. Result

Result manages a window that displays the pathnames of the pages that are returned as the result of a previous query. When *Venn* has submitted a query to *Vax*, the result window is cleared and a message is displayed informing the user that a query is being processed. A message is later received from *Vax* indicating the number of pages found in the query. This information is displayed for the user while *Result* gathers the page names from *Vax*. *Result* will receive a maximum of fifty page names, that are stored in a linked list (extra page names are discarded in the prototype). Once all the page names have been received, they are displayed in the window, that can display a maximum of five page names.

These page names can be scrolled in a manner similar to that of *Keyword_list*. If a page name is selected, it is highlighted and a request is made of *Vax* to display the given page on the Electrohome monitor. Diagram 5.7 shows the main module for *Result*.

```
import( Process_List, Process_Requests )
()
  request   : unsigned
{
  Get_initial_coords();
  Request( DRAW_INIT_RES, QPEL );

  repeat {
    request = Get_Request();
    select(request) {
       case SELECT              : Check_cursor_position();
       case START_MESS          : Display_Working();
       case RECEIVE_NUMBER : Get_Result();
       case RECEIVE_PATH        : Get_New_Name();
    }
  }
}
```

Diagram 5.7 Result main routine

5.2.6. Serial_in

Serial_in monitors the serial line from the remote database system and copies characters into a buffer until a line feed is encountered. When the line feed is detected a request is made of *Vax* to process the data received. *Serial_in* can be considered as a proprietor that works for *Vax*. Diagram 5.8 shows the main module of *Serial_in*.

5.2.7. Vax

Vax handles the protocol between the keyword system and the remote database system. *Vax* passes messages on the serial line to the remote database system when it receives requests from *Venn* and *Result* to do so (*Result* makes requests to display pages). Messages sent to the remote database system are unintentionally echoed back to the keyword system in the same way that keyboard characters are generally echoed to the display device so that a user may see what has been typed in. The echoed messages are not to be considered as results to be passed on to *Result*. *Vax* also receives a terminate request from *Keyboard_in*. When this is received, a message is sent to the remote database system requesting termination of the current session. When verification of the remote termination is received, a request is made of the process manager to destroy all of the processes in the application. Diagram 5.9 shows the main module of *Vax*.

```
import( Process_List, Process_Requests )
()
   ch   : char
{
   Initialize();

   repeat {
      ch = Get_a_char();
      select( ch ) {
         case LF      : { Prepare_to_send();
                          Request( RECEIVE_DATA, VAX );
                        }
         case ' '::'~' :  Buffer_ch();
         \ check for a non-control (printable) character
      }
   }
}
```

Diagram 5.8 Serial_in main routine

```
import( Process_List, Process_Requests )
()
  request : unsigned
{

  repeat {
     request = Get_Request();
     select(request) {
        case RECEIVE_QUERY   : Get_query();
        case DISPLAY_PAGE    : Display_page();
        case RECEIVE_DATA    : Receive_from_serial();
        case TERMINATE       : Terminate();
     }
  }
}
```

Diagram 5.9 Vax main routine

Problems and Prospects

6.1. Appraisal

To date, the prototype system has not been fully completed. The keyword interface functions as described, and the keywords have been generated for each videotex page in the database, but the keyword database query software is still being developed. Therefore, performance testing with the complete system has not been carried out. However, a stub keyword database interface program has been produced to test all other components of the system.

The enforcement of modular design on the system by process structuring allows various components of the system to be modified and tested without affecting the rest of the design. This system can therefore be used as a test-bed for further experiments on the user interface features and configuration.

6.2. Improvements to the interface

Various components of the user interface have been examined for future improvement. Areas of concern include:

- management and manipulation of the keyword list
- display of the query results and their manipulation
- means of giving more information regarding the potential interaction of keywords in the Venn diagram
- improved cursor performance

6.2.1. Keyword list

In the pilot system, the keywords are read from a file on the Port workstation. The specification of which file of keywords to use is made when invoking the keyword system. The words in these files are stored one per line and in alphabetical order. With the current implementation of the system, it is impractical to have a keyword list longer than about sixty to one hundred words, because of the excessive amount of time needed to scroll through the list when searching for words. The window responsible for displaying the keyword list currently displays ten words. Scrolling moves the list by ten words up or down depending on which soft scroll button is picked.

The scrolling of the list is accomplished by erasing the current block and then drawing the next block of words to be displayed. It would be desirable to have a smoothly scrolled list in either direction, but this is not possible with NAPLPS. The only alternative would be to write a coprocessor program to manipulate the video display memory on the board in the desired manner.

If a feature were available to move the user to either the top, bottom or middle of the list, presumably a longer list of words could be handled just as easily. Similarly, a list of any size could be handled with a feature similar to the scroll bar on the Macintosh. Skyvington states that "it is imperative that this powerful scrolling device should exist, in order to enable users to "wander through" lengthy Macintosh documents in the most efficient manner" [Skyvington 84]. The scroll bar allows the user to position a box representing the window anywhere within a vertical bar which represents the document. A feature such as the scroll bar would allow the user to position the keyword display window at any point in the list of keywords. The task of finding a keyword would then be similar to searching for a word in a dictionary or a name in the phone book.

An improved system would allow for user input from a keyboard for pattern matching within the list of words. Various schemes could be employed for unsuccessful matches. The user could be simply told that the search failed. It might be better to place the user in the list at the location where the word would have been located, either alphabetically or phonetically. More elaborate schemes could involve approximate string matching [Johnson 83].

The best alternative would allow user input for pattern searching in combination with a graphical means such as the scroll bar for moving about large keyword lists.

6.2.2. Venn Diagram

In the present implementation of the system, the circles in the Venn diagram are all the same size and appear in the same pattern for all queries. Suppose each keyword in the keyword list obtained from the videotex database had associated with it the percentage use of the word on the pages or some other frequency measure. When keywords are selected from the list, proportionally sized circles could be displayed which would graphically represent the word's frequency in the entire set of pages. This feature would convey more information to the user about the interaction of the words selected. Diagram 6.1 shows two queries. One query uses the regular size circles while the other uses proportionally sized circles and intersection areas. The size of the intersection in the diagram would however not necessarily reflect the number of pages in the intersection of the sets.

Alternatively, a numeric value for the frequency or percentage for a word can be displayed in the Venn diagram, or in the keyword list. However, this technique fails to exploit the graphical aspect of the interface. It may be that this information is difficult to display graphically and easier numerically. If absolute sizes are desirable, then the entire display window could be considered equivalent to the database and circles drawn proportionally to that. This would require some mechanism to enlarge the view for small circles and the need for a corresponding scale. What is potentially misleading to the user in this scenario is the amount of overlap between circles. What would be ideal would be the ability to show the

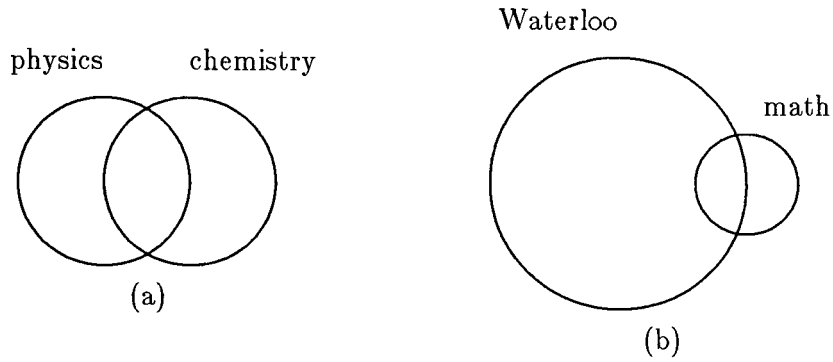


Diagram 6.1 Proportional circles

proper overlap, but this is not known beforehand in a post-coordinate keyword system. The overlap region would have to be calculated by actually executing the query each time for each pair or group of circles in question. However, a reasonable approximation could be calculated by examining the inverted lists. Another simple approach could have the circle's size drawn on a scale from 1 to 10 based on the distinctiveness of the keyword. The intended purpose of giving this extra "size" information is to give the user a better idea about how many pages would be retrieved for the given query before processing.

For simplicity, the prototype system uses "X"s to indicate which regions in the Venn diagram were selected. NAPLPS is capable of filling in closed figures, such as circles, with colours and patterns. This would have been desirable but difficult to implement because of the unusual regions produced by the overlapping circles.

6.2.3. Output

In the prototype system, the Unix pathnames for the pages are returned for display in the result window. The user selects from these file names which pages are to be displayed. Because of the hierarchical structure of the database of pages, the path names are fairly informative. However, in general this is unacceptable. Some other type of naming scheme is needed here but it is not clear what that should be.

The problem is differentiation between the pages retrieved. If a query returns less than ten pages, then a numbering scheme could be employed and this method could be just as good as any other. Possibly representative keywords for that page could be used to identify it. Even miniatures of the actual pages could be used [Feiner 82].

If more than some threshold number of pages were returned, then the user would probably want to refine the query to reduce the amount of pages found. Cooper suggests that output from a boolean retrieval language should be ranked in some way so that the user can examine the more promising documents first [Cooper 82] . What is meant by "promising" could differ significantly depending on whose perspective is used: the user's or the information provider's. Ranked output would be very

attractive especially when the number of pages retrieved is larger than some comfortable threshold. Cooper suggests that "Several of the drawbacks of conventional information retrieval systems can be overcome by a design approach in which queries consist of sets of terms, either unweighted or weighted with subjective term precision estimates, and retrieval outputs are ranked by probability of usefulness estimated in accordance with the so-called "maximum entropy principle"". This idea could be implemented in the system if the keywords were weighted with a best guess as to what fraction of the pages containing that term would be useful if retrieved. If no weighting were given, then equal weighting could be assumed. Including this feature would probably require another input window for assigning the weights to the keywords or some means of selecting a word in the Venn diagram and attaching a weight. In either case, the trade-off between interface complication and the ability to have ranked output would have to be examined.

When a query is invoked, the only indication the user gets that the query is being processed is a "working" message in the display window after the display window has been cleared. It would be very informative and reassuring to have some type of indication as to how the query was progressing. Myers has shown that percent-done progress indicators appear to be an important aid to users in various ways and that systems with progress indicators are preferred [Myers 85]. An example of a percent-done indicator would be a graphical representation of an hourglass or thermometer which would show the percentage of the task completed by having the sand fall or the mercury rise. These graphical indicators help novices feel better about the system by showing that a command has been accepted and the task is progressing successfully. They also provide experienced users with enough information to estimate completion times and therefore allow for better planning and use of time in a multi-process environment. Since the keyword system has been developed in a multi-process environment, the user might form another query while the existing one is being processed. However, a user would typically wait for the response to a query before continuing. If there were a convenient way to save queries for later recall and modification, then the user could take advantage of the parallelism that the system has to offer, and would probably want a percent-done indicator for each query.

6.2.4. Cursor

The current cursor implementation, as discussed in Section 3.2.3, is the best of the alternatives examined, but still does not move well. As mentioned previously in Section 3.1, the Quickpel board has 16K of RAM for coprocessor programs. Coprocessor programs must be written in 8088 assembler language and downloaded to the Quickpel for execution. (See the Quickpel user's guide for details [Electrohome 84].) A test coprocessor program has been written for the Quickpel board to draw and control a personalized cursor different from those discussed previously. A second test program on the Port workstation monitors the movement of the mouse and sends control bytes to the program on the Quickpel board.

This cursor behaves much more nicely and can be varied in shape, size, and colour. However, the test program only moves the cursor in four directions and does not have the flexibility to position it anywhere on the screen in one command as with the NAPLPS drawing point. With further modification and testing, these programs could replace and improve the appearance of the cursor in the next version of the keyword system.

6.2.5. Remote Database Communication

Before starting the prototype keyword system, the user must first use the terminal icon from the Port interface to access the remote keyword and videotex database. Once the user has logged onto the remote system and has started the keyword database application program the keyword system is started on the Port workstation. After quitting the terminal icon, the serial line is still connected and can be used by the keyword system to transmit data to the remote database system. This method was used so that software duplicating the actions of the terminal process would not have to be developed.

The remote keyword database interface is a Unix "csh" program that expects commands and queries from the standard input and writes the query results to the standard output. Queries are in the format generated by the Venn manager as discussed in Section 5.6. The Vax manager adds a carriage return and line feed to the query when sending it to the remote database system. Because of this arrangement, the Vax manager must be able to receive and discard the echoed query sent to the remote system. The session with the remote database is terminated when the unique terminate command is sent and acknowledged. No other protocol system is used and for test purposes this arrangement has been satisfactory. If this system should go past the prototype stage, then a proper protocol should be used to transmit and receive data from the remote database system.

6.3. Conclusion

Keyword interfaces have potential for information retrieval systems if queries can be expressed in a more natural means than boolean expressions. Thus, keyword systems based on Venn diagrams may be useable by infrequent users.

Some tests need to be performed with the prototype keyword system once the query processing software has been completed to evaluate its feasibility. Also, the modularity of the design of the keyword system facilitates experimentation with different front ends to a videotex database. Therefore various tests should be conducted to determine which features are desirable and conducive to retrieval.

Basic rules or guidelines for system design using anthropomorphic process models need to be identified and evaluated. The use of the process manager model in the keyword system design and implementation was successful and should be explored more thoroughly.

References

[Ball 1981]
User Interfaces for Future Videotex Systems
A.J.S. Ball and J. Gecsei
Publication #412, Départment d'informatique et de recherche opérationelle, Université de Montréal, May 1981

[Baron 69]
A Note on the Historical Development of Logic Diagrams
M. E. Baron
The Mathematical Gazette, 53 (May 1969), pp. 113-125

[Bochmann 1982]
Keyword Access in Telidon: An Experiment
G.V. Bochmann, J. Gecsei and E. Lin
Videotex '82, New York, N.Y. June 28-30, 1982, pp. 321-332

[Booth 84]
Anthropomorphic Programming
K. Booth, J. Schaeffer, W.M. Gentleman
University of Waterloo, Department of Computer Science
Technical report CS-84-47, February, 1984.

[Bradley 84]
Assembly Language Programming for the IBM Personal Computer
David J. Bradley
Prentice-Hall, Inc., 1984

[Brenner 79]
Indexing in Perspective
E.H. Brenner
Unesco/Unisist, 1979.

[Cargill 79]
A View of Source Text for Diversely Configurable Software
T. A. Cargill
University of Waterloo, Department of Computer Science
Technical report CS-79-28, 1979.

[Cooper 83]
Exploiting the Maximum Entropy Principle to Increase Retrieval Effectiveness
William S. Cooper
Journal of the American Society for Information Science, 34(1):31-39; 1983

[CSA 83]
Videotex/Teletext Presentation Level Protocol Syntax
Canadian Standards Association, Rexdale, Ontario, December 1983.

[Dict of Sci Bio 76]
Dictionary of Scientific Biography
Volume 13, pp 611-613
Charles Scribner's Sons, New York, 1976.

[DOC 81a]
Telidon Behavioural Reasearch 2: The Design of Videotex Tree Indexes
Department of Communications, Ottawa, May 1981.

[Electrohome 84]
Quickpel User's Guide
Electrohome Limited, Version 1.0, June 1984.

[Feiner 82]
An Experimental System for Creating and Presenting Interactive Graphical Documents.
Steven Feiner, Sandor Nagy, and Andries van Dam
ACM Transactions on Graphics, Vol. 1, January 1982, pp. 59-77.

[Furnas 82]
Statistical Semantics: Analysis of the Potential Performance of Keyword Information Systems.
G.W. Furnas, T.K. Landauer, L.M. Gomez, S.T. Dumais
1982

[Gecsei 1983]
The Architecture of Videotex Systems
Jan Gecsei
Prentice-Hall, 1983

[Intel 83]
iAPX 86/88, 186/188 User's Manual
Intel Corporation, 1983

[Johnson 83]
Formal Models for String Similarity
J. Howard Johnson
Ph.D. dissertation, University of Waterloo, Department of Computer Science, Technical report CS-83-32, November 1983

[Malcolm 85]
Real Time Programming — class notes from CS 652
M. A. Malcolm
University of Waterloo, Department of Computer Science
Spring term 1985

[Michard 82]
Graphical presentation of boolean expressions in a database query language: design notes and an ergonomic evaluation
A. Michard
Behavior and Information Technology, 1982, Vol. 1, No. 3, pp. 279-288

[Mills 81]
Telidon Behavioural Reasearch 3: A study of the human response to pictorial representations on Telidon
Micheal L. Mills
Department of Communications, Ottawa, May 1981.

[Myers 85]
The importance of Percent-Done Progress Indicators for Computer-Human Interfaces.
Brad A. Myers
Proceedings of the 1985 ACM SIGCHI, April 85, pp. 11-17

[Orsnaes 1982]
User Reactions to Keyword Access Videotex
Jorn Orsnaes
Videotex '82, New York, N.Y., June 28-30, 1982, pp. 315-320

[Raymond 84]
Personal Data Structuring in Videotex
Darrell Raymond
Masters thesis, University of Waterloo, Department of Computer Science
Technical report CS-84-7, Feburary 1984

[Raymond 85]
Videotex Fact Retrieval
Darrell Raymond, Frank Tompa
University of Waterloo, Department of Computer Science
unpublished technical report

[Raymond 86]
Structuring Processes with Managers
Darrell Raymond
University of Waterloo, Department of Computer Science
unpublished technical report

[Simonds 84]
Database Limitations and Online Catalogs
Michael J. Simonds
Library Journal, Feburary 15, 1984, pp. 329-330

[Skyvington 84]
Mastering Your Macintosh
William Skyvington
Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp. 25-31

[Stewart 79]
Prestel - How Usable is it?
T. Stewart
Proceedings of the Conference in Human Aspects of Telecommunication: Individual and Social Consequences,
Munich, Springer-Verlag, October 1979, pp. 107-117

[Stewart 80]
Human Factors in Videotex
T. Stewart
Proceedings of the 4th International Online Information Meeting,
London 9-11, December 1980, pp. 87-95

[Vigil 83]
The Psychology of Online Searching
Peter J. Vigil
Journal of the American Society for Information Science, Vol. 34, No.4,
July 1983, pp. 281-287

[Wason 72]
Psychology of Structure and Reasoning: Structure and Content.
P.C. Wason and P.D. Johnson-Laird
Harvard University Press, Cambridge, MA.,1972

[Webster 83]
Webster's New World Dictionary
Simon & Schuster, Inc., 1983

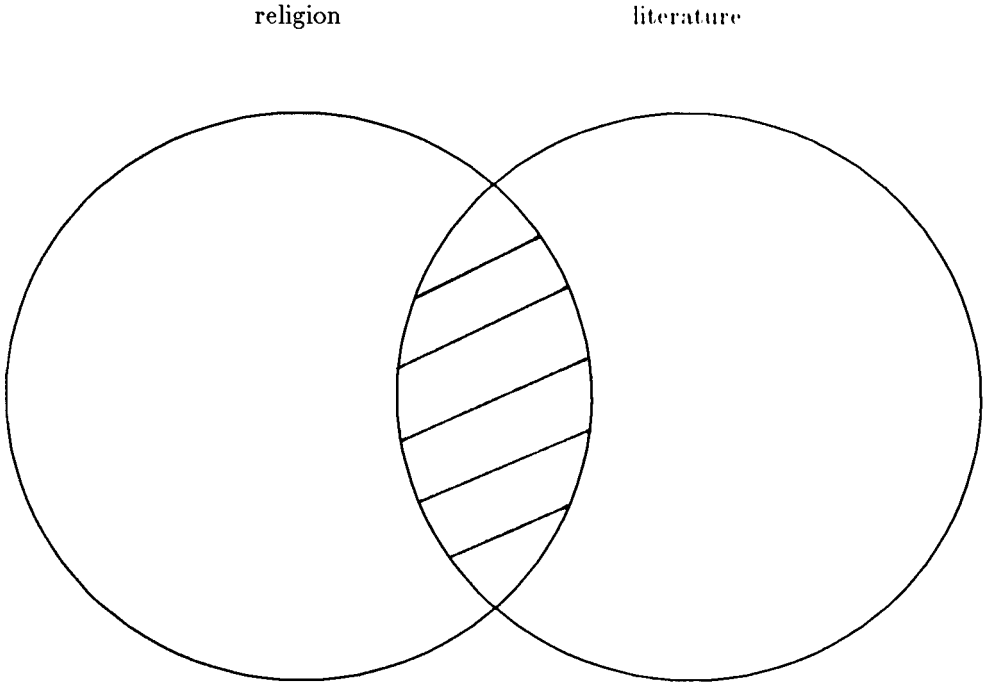
[WMI 86a]
Waterloo Port's Port Development System Manual
Waterloo Microsystems Inc., Waterloo, Ontario, 1986

[WMI 86b]
Waterloo Port User's Guide
Waterloo Microsystems Inc., Waterloo, Ontario, 1986

Appendix A.1 : Experimental Instructions

In this task you will be given some pages with Venn diagrams on them. You will be asked to describe the areas that have been shaded in. Your description should enable someone else to identify the given areas on a similar Venn diagram which does not have the shaded portions.

The following is an example :



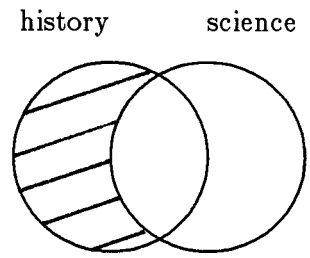
Possible descriptions :

1) religious literature
2) writings by holy men: bible, koran.

Appendix A.2 : Data Collected from Experiment

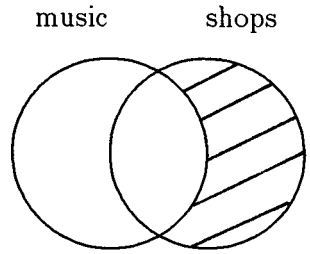
Each page includes the test condition and results from the subjects for that condition. Corrections in spelling or grammar *have not* been made to the results obtained from the subjects. Those lines beginning with an ⊗ denote responses which were considered to be incorrect.

Condition 1 (general)



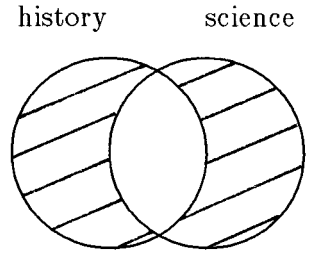
1. history only
2. history not including the sciences (ie. not science)
3. non-scientific history (ie. the battle of Waterloo)
4. history that is not related to science
5. pure historians
6. historical but not scientific (not dealing with science)
7. Descr: (1) history of fictional literature (2) history of non-scientific discoveries (3) history of chronological facts/occurences
8. A pure historian not concerned with scientific discovery
9. non-scientific historic occurences
10. Examples: World War 1 & 2, yesterday, birth of Christ
11. the area pertaining only to history; no scientific data or information included, nor any overlap or combination of scientific and historical information. ie history of WW's; history of English literature
12. history eliminating the history of science
13. all of history that is not dealing with science
14. history, no science
15. all parts of history that doesn't include references to science
16. That portion of our history which does not involve scientific advancement or discovery

Condition 1 (specific)



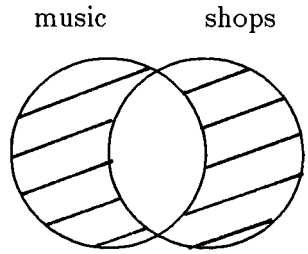
17. All stores that don't play music
18. those things (ie. retail stores, producers) which are not involved in the sale or production of musical instruments, music books, and other things associated with music.
19. shops not selling anything related to music
20. shops which are not in the music industry ie. they don't sell music, or musical instruments etc.
21. shops not selling musical things (instruments, etc) grocery, clothing,shoe stores books,...
22. shops that don't sell music
23. shops that don't sell instruments or any music related articles ie records
24. place where you would buy something not related to music
25. stores other than music stores ie not record stores etc.
26. shopstores that don't sell musical equipement of any type
27. There are no music shops
28. All shops that have nothing to do with music. ie the portion of shops which does not intersect with music
29. all shops which don't sell music
30. shops that don't sell or play music
31. shops only, no music whatsoever
32. non-musical shops, pure shops, no musical instruments sold

Condition 2 (general)



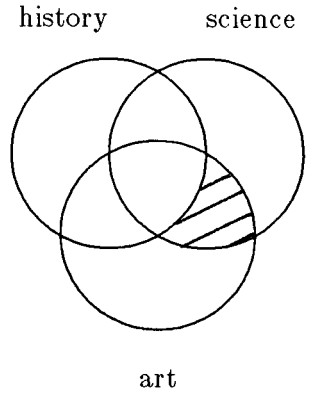
1. ⊗ omit history of science
2. history or science, but excluding their overlap ie Not(science and history)
3. ⊗ the mutually exclusive aspects of science and history (eg We don't want to hear about Galileo's invention of the telescope)
4. all relating to history and all relating to science but NOT anything relating to both history and science
5. ⊗ Not interested in historical science.
6. history or science but not history and science
7. ⊗ Descr: (1) Reviews of actual occurrances and scientific happenings.
8. ⊗ The study of history and science but not the history of science or the science of history
9. ⊗ non-scientific history and non-historic science
10. ⊗ Examples: type 1: birth of Chirst; type 2: computers, Exclude : space shuttle disaster
11. areas including historical and scientific info. separately. history contains no scientific info and likewise, the scientific info does not contain any historical info. (eg history --> history of Canadian politics, science --> the anatomy of the human body)
12. ⊗ everything but the history of science
13. all of history that is not dealing with science plus all of science that does not deal with history.
14. history or science no historical science
15. ⊗ everything in history and science except historical science.
16. Recorded history and scientific knowledge as it exists today. The history of scientific advancement is excluded.

Condition 2 (specific)



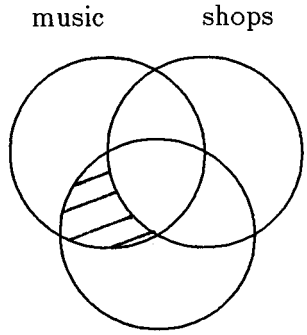
17. ⊗ all music except music played in shops and all shops where music isn't played
18. ⊗ the things related to music which have nothing to do with shops AND the things related to shops which have nothing to do with music.
19. ⊗ Everything about music and shops but not about music shops
20. All music which is not sold in a shop or all shops that do no sell music OR everthing BUT music sold in a shop.
21. shops selling either music supplies or other things, but not both. ie instrument store, record shop.
22. ⊗ Anything but music shops
23. All the shops and all the music but not the music shops (shops that sell music related articles ie records, instruments, etc)
24. Areas which deal only with music and only with shops, NOT with the intersection of the two.
25. ⊗ All shops other than music shops and all aspects of music other than music shops
26. ⊗ not musical shops
27. ⊗ There are some shops that do not deal with music in any way.
28. those portions of music and shops that do not intersect with each other.
29. anything associated with either music or shops, but not music shops.
30. ⊗ the opposite of shops that sell or play music
31. shops and music except for musical shops
32. Not music shops; music or shops but not both; anything but musical shops

Condition 3 (general)



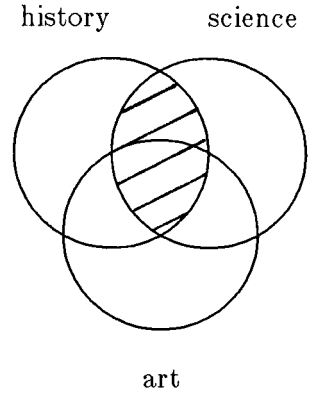
17. students taking science and art but not history
18. the things which are related to science AND art excluding the things which are related to art and science and history
19. All information on art and science that is NOT of historical nature
20. all art about science which is not historical
21. ⊗ scientific and technical drawing/modelling, illustrated book on anatomy.
22. looking at science and art but ignoring history
23. The things in common between art and science that have nothing to do with history
24. intersection of art and science - no involvement of history
25. the overlapping of science and art without history
26. ⊗ bag pipes are a science and an art
27. ⊗ scientific art
28. the portion of intersection between art and science that does not intersect with history
29. In a university context, students taking art and science but not history
30. topic that contains both science and art but not history
31. scientific art having nothing to do with history
32. non-historical scientific art; modern computer art/graphics

Condition 3 (specific)



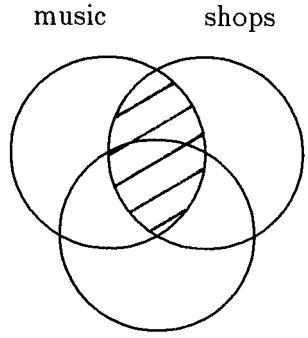
1. music transportation without shops
2. ⊗ (music and transportation) or (not shops)
3. ⊗ wierd --> driving music that a mechanic wouldn't play
4. ⊗ a mode of transportation on which you hear music
5. ⊗ A van with Alpine speakers. Not for sale.
6. intersection only of transportation and music, but not shops
7. ⊗ Descr (1) car-radio
8. ⊗ listen to the radio in the car
9. ⊗ car stereo
10. ⊗ Examples : merry-go-round, car with a radio
11. area of interest: overlap between music and transportation ONLY (ie exclude any ideas about shops, stores,etc.) eg. info on car stereos, walkmans, etc.
12. Musical transportation, no shops
13. all of music that deals with transportation excluding anything to do with shops but no more
14. musical transportation - no shops
15. ⊗ places where one must travel to hear music,write,record music not including travelling to buy music at a shop
16. ⊗ Music you listen to on your car stereo while driving or on your walkman while walking.

Condition 4 (general)



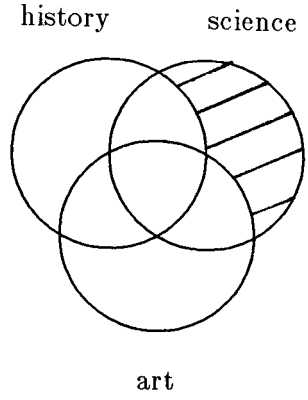
17. students taking history, science and art OR just history and science
18. the things related to history and science
19. Everything about the history of science
20. all history about science
21. ⊗ Books: on evolution of man from cave art; illustrated book on history of a science topic (eg discovery of insulin, electricity, etc.) How art has evolved in time. Leonardo da Vinci.
22. ⊗ The history of art and science
23. ⊗ The part of art that only deals with the intersection (part in common) of history and science
24. intersection of history and science
25. the overlapping of history and science
26. the printing press made a contribution to history, science and art, and is part of history and science
27. historical science that may contain art
28. the intersection between all three (history,science and art) plus the intersection between history and science which does not intersect with art.
29. In a university context, any student taking both science and history, regardless of whether he/she is taking art.
30. topic involves history and science, not just one, and may also involve art.
31. scientific history
32. historical science, possibly artistic; scientific history including that relating to art

Condition 4 (specific)



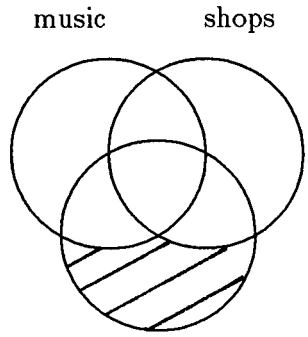
1. music shops and music shops with transportation
2. (music and shops and transportation) or (music and shops)
3. music shops eg. stereo stores, music stores, dance school (is sort of "music shop") - portable music store ? (for transportation) - music played in a repair shop.
4. ⊗ music shops accessible by some mode of transportation
5. a music store
6. intersection of music and shops
7. ⊗ Descr: (1) Record stores and records-on-wheels companies
8. the tapes in the car were purchased in a shop
9. ⊗ car radio stores
10. Example: A&A records, Sam the Record Man
11. ⊗ are a of interest: info pertaining to music in stores and how transportation has any affect on this relationship.
12. music store
13. all of music that deals with shops but no more
14. ⊗ transportation within musical shops and musical shops
15. ⊗ shops where you can buy music, transportation does not play a role.
16. music stores which sell records etc plus those music stores which sell portable music stations such as walkmans and ghetto blasters

Condition 5 (general)



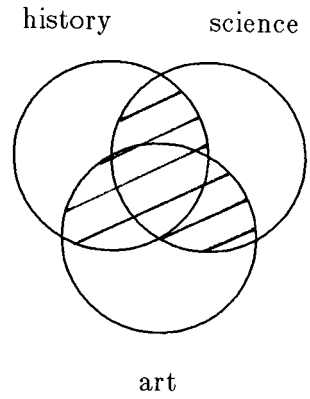
17. all students taking only science (no history or art)
18. all the things related to science excluding those things which are also related to history OR art
19. Everything on science which is not historical in nature and does not have to do with art
20. all science which has nothing to do with art or history
21. chemistry formulas; physical properties of matter; lab experiments
22. looking at science and how it is distinct form history and art
23. the part of science that does not overlap any of history or of art
24. area of science not affected by any intersections; ONLY science
25. the independent aspects of science not overlapped with history or art
26. star trek is science fiction not art or history
27. science that deals with neither history or art
28. the part of science which does not intersect with any part of art or any part of history
29. In a university context, students who take science but who do not take history and/or art.
30. topic is only science, does not contain either history or art
31. science only, no art, no history
32. non-historical science which is not considered art; pure science

Condition 5 (specific)



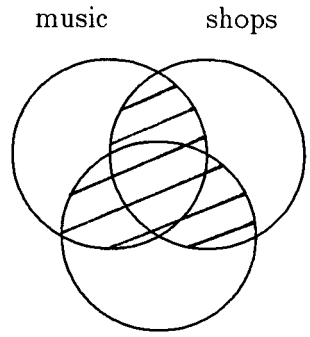
1. transportation only
2. ⊗ transportation or not music or not shops
3. hard to find an example of this. - travelling by train (no music but the train still has to be serviced) - travelling by foot.
4. a mode of transportation on which you would NOT hear any music and NOT pass by any shops
5. strictly transportation
6. transportation not dealing with music or shops
7. Descr: (1) tranquil canoe trip (2) bicycle ride (3) personal car - no noise ride
8. The train or bus are queit.
9. bus
10. Examples: bicycle, skateboard, windsurfer
11. info pertaining only to transportation (ie. exclude any ideas about music and shops) eg. info about community transit, future developments in transport.
12. ⊗ no music and no shops
13. all of transportation that doesn't deal with music of shops and nothing else
14. transportation, no music or shops
15. places to go using transportation where no music and no shops are present.
16. driving, walking or taking the bus.

Condition 6 (general)



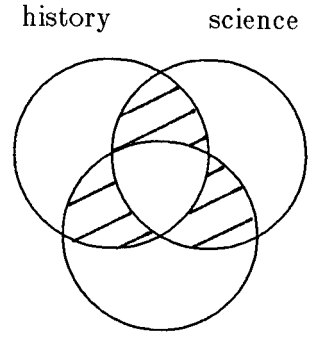
17. all students taking at least two of history, art and science
18. all things which are related to history and science OR all things which are related to history and art OR all things which are related to science and art
19. Everything that has to do with any two of science, art and history
20. all history about science OR all history about art OR all art about science
21. ⊗ art history; technical drawings; evolution of science; evolution of art; illustrated book on history and/or science; (scientists, historians,artists) any combo thereof.
22. ⊗ Examining the similarities between every subclass of history, art and science
23. The parts of history, science and art that overlap at least one other of the mentioned categories.
24. areas shown by the intersections of : history and art; science and art; history and science
25. ⊗ the overlapping of history, art and science
26. at least two of history, science and art
27. any art, history or science that deals with one or both of the others
28. all of the intersecting areas between the three groups. Ie. all of the area other than the part of each of history, science and art which does not intersect with any part of the other two. The shaded area looks like a three leaf clover.
29. In a university context, students taking any two or three of science, art and history
30. topic contains at least two of history, science and art.
31. scientific history and art history and scientific art
32. historical art, scientific art or historical science including items which are scientific, historical and artistic in nature.

Condition 6 (specific)



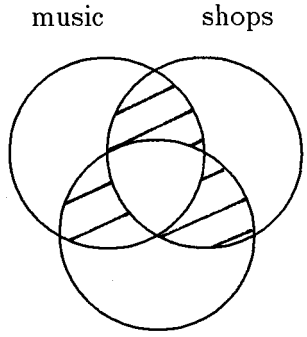
1. music shops with transportation, shops with transportation, music shops, music with transportation
2. (music and shops) or (shops and transportation) or (music and transportation)
3. ⊗ none of these are included { "music to sleep by",walking, computer repair shop that doesn't play music} In real life, the non-shaded areas are very small. (mechanics shop, music about the road, music repair shop.)
4. ⊗ a mode of transportation that will bring you to the shop where there is music playing both in the shop and while you are travelling.
5. ⊗ Employment at "Records on Wheels" includes driving
6. intersection of the following : (1) transportation and music and shops and (2) music and shops and (3) transportation and shops and (4) music and transportation
7. ⊗ Descr: (1) musical instrument delivery van
8. ⊗ a band travels to many places to play music, they also travel to various stores, some are music stores
9. ⊗ store selling only musical equipement for automobiles
10. ⊗ Examples: (1) car with radio (2) A&A Records (3) escalators in Eatons, elevators in Sears
11. info including any combination of the three variables or even just a combination of only two. eg music is stores and transportation or music is stores to create atmosphere or transportation and shops
12. music and shops , shops and transportation , music and transportation
13. anything dealing with at least two of music , shops , and transportation
14. ⊗ music , shops and transportation all equal where they connect
15. Places to go using transportation where one can buy music in shops . Also places to travel for shopping. Also travel to hear music and also music shops in general.
16. Contained in this group are music stores, bike and car dealers, and music stores which specialize in offering music for the traveller. Walkmans, car stereos ghetto blasters and tapes to play constitute the stock of the latter store.

Condition 7 (general)



17. all students taking at least two of but no more than two of history, art and science
18. those things which are related to history and science OR those things which are related to history and art OR those things which are related to science and art but excluding those things which are related to history and science and art (ie related to all three)
19. Everything that has to do with any two out of history, science and art, but not with the three of them.
20. all history about science OR all history about art OR all art about science BUT NOT any historical art about science
21. ⊗ art history; history of science; scientific methods applied to history research; scientific drawing; how to draw.
22. - NULL -
23. The sections of two overlapping circles but not three overlapping circles.
24. get area of intersections of art and history, art and science, history and science and subtract the area in which art, history and science all intersect.
25. ⊗ the overlapping of history, art, and science other than the area where all three overlap.
26. ⊗ two of history, science and art
27. matters dealing with any pair of art, history or science
28. the intersection bewteen history and art, plus the intersection between art and science, plus the intersection between science and history with the intersection between all three at once removed. The shaded area looks like a three leafed flower with the centre removed.
29. In a university context, students who take any two of science, history and art but who do not take all three.
30. topic contains exactly two of history, science and art but not all three.
31. scientific history and art history and scientific art except for scientific art history
32. historical art, scientific art, historical science but excluding items which are considered historical scientific art

Condition 7 (specific)



1. ⊗ music shops without transportation, shops with transportation without music and music transportation without shops
2. (music and shops) or (shops and transportation) or (transportation and music) or not (music and shops and transportation)
3. same as previous page except things like a mechanics shop that plays music are not included
4. ⊗ there is music playing in the shop and on your mode of transportation that is bringing you to the shop but the music is not the same in both places.
5. ⊗ I'm a taxi driver who loves to frequent music stores. But I'll never work there.
6. ⊗ intersection of: (music and shops) and (transportation and shops) and (transportation and music) but not (transportation,music and shops)
7. ⊗ Descr: car radio or radio shop or shop car.
8. ⊗ I like to travel and listen to music. When I drive to the store I don't listen to music, but I like it playing in the store.
9. - NULL -
10. - NULL -
11. info pertaining to the overlap of only two variables eg (music and transportation) or (shops and transportation) or (music and shops). Each excludes the third variable
12. music and shops, shops and transportation, transportation and music, without intersection of music, shops and transportation
13. anything dealing with only two of music,shops and transportation. (Does not include the intersection of all three)
14. ⊗ music, shops, and transportation all equal but only two shaded areas within each circle
15. All music-shops where transportation is not needed Musical experiences requiring transportation but not at any shops. All shops where transportation is required not including music-shops
16. music stores, motorcycle and car dealers and portable music stations belong to this group. However, stores which sell portable music stations are not included.